Java Programming (Oracle)

6661 36 weeks

Table of Contents

Acknowledgments ......................................................................................................................................... 2
Course Description........................................................................................................................................ 3
Task Essentials Table .................................................................................................................................... 3
Curriculum Framework ................................................................................................................................. 8
Comparing Java and PL/SQL Programming ................................................................................................ 9
Learning the Fundamentals of Java as an Object-Oriented Programming Language .............................. 11
Distinguishing Key Elements of Java ........................................................................................................... 16
Controlling Program Flow ............................................................................................................................ 23
Introducing Classes, Objects, and Methods ................................................................................................. 30
Using Data Types and Operators ................................................................................................................ 36
Understanding Methods and Classes ........................................................................................................... 38
Using Inheritance ........................................................................................................................................ 40
Using Packages and Interfaces .................................................................................................................... 48
Handling Exceptions .................................................................................................................................... 51
Exploring Database Input/Output ................................................................................................................ 58
Understanding Multithreaded Programming Support ................................................................................ 62
Creating Applets, Events, and Other Topics ................................................................................................. 66
Using Databases with Java .......................................................................................................................... 71
Preparing for Industry Certification ............................................................................................................ 78
SOL Correlation by Task ............................................................................................................................... 81
Instructional Resources for Java .................................................................................................................. 85
Cyber Security and Cyber Forensics Infusion Units ..................................................................................... 86
Entrepreneurship Infusion Units .................................................................................................................. 86
Appendix: Credentials, Course Sequences, and Career Cluster Information ............................................. 87
Acknowledgments

The components of this instructional framework were developed by the following curriculum development panelists:

Jill Anderson, Instructor, Matoaca High School Center for Information Technology, Chesterfield County Public Schools
Frank Baber, Director, Oracle Cloud Solution Hubs Developer Innovation Team, Oracle, Reston
Gail Drake, Instructor, Battlefield High School, Prince William County Public Schools
Denise Hobbs, Regional Director, North America, Oracle, Albuquerque, New Mexico
Lisa Palombo, Senior Curriculum Developer, Oracle, St. George, Utah
Brian Thomas, Instructor, James Wood High School, Frederick County Public Schools

Correlations to the Virginia Standards of Learning were reviewed and updated by:

Leslie R. Bowers, English Teacher (ret.), Newport News Public Schools
Vickie L. Inge, Mathematics Committee Member, Virginia Mathematics and Science Coalition
Anne F. Markwith, New Teacher Mentor (Science), Gloucester County Public Schools
Michael Nagy, Social Studies Department Chair, Rustburg High School, Campbell County Public Schools

The framework was edited and produced by the CTE Resource Center:

Nathan K. Pope, Writer/Editor
Kevin P. Reilly, Administrative Coordinator

Virginia Department of Education Staff

Judith P. Sams, Specialist, Business and Information Technology and Related Clusters
Dr. Tricia S. Jacobs, CTE Coordinator of Curriculum and Instruction
Dr. David S. Eshelman, Director, Workforce Development and Initiatives
George R. Willcox, Director, Operations and Accountability

Office of Career, Technical, and Adult Education
Virginia Department of Education

Copyright © 2019
Course Description

Suggested Grade Level: 11 or 12  
Prerequisite: 6660

Students study Java, perhaps the most widely used object-oriented, class-based, general-purpose programming language, to create and manipulate database objects and applications. Instruction will emphasize preparation for industry certification.

Task Essentials Table

- Tasks/competencies designated by plus icons (➕) in the left-hand column(s) are essential  
- Tasks/competencies designated by empty-circle icons (🔍) are optional  
- Tasks/competencies designated by minus icons (➖) are omitted  
- Tasks marked with an asterisk (*) are sensitive.

<table>
<thead>
<tr>
<th>Task Number</th>
<th>6661</th>
<th>Tasks/Competencies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Comparing Java and PL/SQL Programming</td>
</tr>
<tr>
<td>39</td>
<td>➕</td>
<td>Compare PL/SQL and Java.</td>
</tr>
<tr>
<td>40</td>
<td>➕</td>
<td>Identify the advantages of using PL/SQL in database application development.</td>
</tr>
<tr>
<td>41</td>
<td>➕</td>
<td>Identify the advantages of using Java in database application development.</td>
</tr>
</tbody>
</table>

Learning the Fundamentals of Java as an Object-Oriented Programming Language

<table>
<thead>
<tr>
<th>Task Number</th>
<th>6661</th>
<th>Tasks/Competencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>➕</td>
<td>Describe Java's place in computer-language history.</td>
</tr>
<tr>
<td>43</td>
<td>➕</td>
<td>Explain the fundamentals of object-oriented programming.</td>
</tr>
<tr>
<td>44</td>
<td>➕</td>
<td>Describe the general form of a Java program, including variables, program flow control, main, and methods statements.</td>
</tr>
<tr>
<td>45</td>
<td>➕</td>
<td>Create and compile a block of code.</td>
</tr>
<tr>
<td>46</td>
<td>➕</td>
<td>Identify Java keywords.</td>
</tr>
<tr>
<td>47</td>
<td>➕</td>
<td>Describe the standard Java naming conventions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>48</td>
<td>+</td>
<td>Identify the key components of the Software Development Kit (SDK).</td>
</tr>
<tr>
<td>49</td>
<td>+</td>
<td>Describe the JVM.</td>
</tr>
<tr>
<td><strong>Distinguishing Key Elements of Java</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>+</td>
<td>Use Java's primitive data types.</td>
</tr>
<tr>
<td>51</td>
<td>+</td>
<td>Initialize and assign variables.</td>
</tr>
<tr>
<td>52</td>
<td>+</td>
<td>Describe the scope rules of a variable.</td>
</tr>
<tr>
<td>53</td>
<td>+</td>
<td>Apply type conversion (casting) in expressions.</td>
</tr>
<tr>
<td>54</td>
<td>+</td>
<td>Use the arithmetic operators.</td>
</tr>
<tr>
<td>55</td>
<td>+</td>
<td>Compare relational and logical operators.</td>
</tr>
<tr>
<td>56</td>
<td></td>
<td>Input characters from the keyboard.</td>
</tr>
<tr>
<td>57</td>
<td>+</td>
<td>Navigate the Java API.</td>
</tr>
<tr>
<td>58</td>
<td>+</td>
<td>Build and execute Java applications, using a development tool.</td>
</tr>
<tr>
<td>59</td>
<td>+</td>
<td>Modify Java application source code.</td>
</tr>
<tr>
<td><strong>Controlling Program Flow</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>+</td>
<td>Identify situations in which to use control statements.</td>
</tr>
<tr>
<td>61</td>
<td>+</td>
<td>Use if-then and if-then-else statements.</td>
</tr>
<tr>
<td>62</td>
<td>+</td>
<td>Use switch statements.</td>
</tr>
<tr>
<td>63</td>
<td>+</td>
<td>Use for loop statements.</td>
</tr>
<tr>
<td>64</td>
<td>+</td>
<td>Use the while loop.</td>
</tr>
<tr>
<td>65</td>
<td>+</td>
<td>Use the do-while loop.</td>
</tr>
<tr>
<td>66</td>
<td>+</td>
<td>Use break and continue statements effectively.</td>
</tr>
<tr>
<td><strong>Introducing Classes, Objects, and Methods</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>+</td>
<td>Identify class and method definitions.</td>
</tr>
<tr>
<td>68</td>
<td>+</td>
<td>Differentiate between classes and objects.</td>
</tr>
<tr>
<td>69</td>
<td>♦</td>
<td>Describe an object and its operations (or methods).</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>70</td>
<td>♦</td>
<td>Create a method, and return a value from a method.</td>
</tr>
<tr>
<td>71</td>
<td>♦</td>
<td>Use parameters in a method.</td>
</tr>
<tr>
<td>72</td>
<td>♦</td>
<td>Add a constructor to a class.</td>
</tr>
<tr>
<td>73</td>
<td>♦</td>
<td>Overload a constructor.</td>
</tr>
<tr>
<td>74</td>
<td>♦</td>
<td>Describe garbage collection and the finalize ( ) method.</td>
</tr>
<tr>
<td>75</td>
<td>♦</td>
<td>Apply the this reference.</td>
</tr>
<tr>
<td>76</td>
<td>♦</td>
<td>Explain the purpose and importance of the Object class.</td>
</tr>
</tbody>
</table>

**Using Data Types and Operators**

| 77 | ♦ | Use single and multidimensional arrays. |
| 78 | ♦ | Use various formats to initialize arrays. |
| 79 | ♦ | Apply the bitwise operators. |
| 80 | ♦ | Create objects of type String and utilize their methods. |

**Understanding Methods and Classes**

| 81 | ♦ | Define the public- and private-access modifiers. |
| 82 | ♦ | Pass and return objects/primitive parameters to methods. |
| 83 | ♦ | Use class variables and class methods. |
| 84 | ♦ | Apply recursive methods. |
| 85 | ♦ | Create and use recursive methods and variables. |
| 86 | ♦ | Describe nested and inner classes. |

**Using Inheritance**

| 87 | ♦ | Use inheritance to define new classes. |
| 88 | ♦ | Describe the difference between a superclass and a subclass. |
| 89 | ♦ | Explain how inheritance affects member access. |
| 90 | 🟢 | Use *super* to call superclass constructors and members. |
| 91 | 🟢 | Create a multilevel class hierarchy. |
| 92 | 🟢 | Explain cases when constructors are called in a class hierarchy. |
| 93 | 🟢 | Apply superclass references to subclass objects. |
| 94 | 🟢 | Demonstrate how to override methods. |
| 95 | 🟢 | Create abstract methods and classes. |
| 96 | 🟢 | Use the keyword *final* to prevent inheritance. |
| 97 | 🟢 | Create and implement an interface, utilizing its variables. |
| 98 | 🟢 | Extend an interface. |
| 99 | 🟢 | Create and import a package. |
| 100 | 🟢 | Describe how a package affects access to its programs. |
| 101 | 🟢 | Explain the purpose of the Thread class and the Runnable interface. |
| 102 | 🟢 | Explain the purpose of exception handling. |
| 103 | 🟢 | Write code to demonstrate how *try* and *catch* work together to handle an exception. |
| 104 | 🟢 | Create an exception handler. |
| 105 | 🟢 | Identify the consequences of an unhandled exception. |
| 106 | 🟢 | Use multiple *catch* statements. |
| 107 | 🟢 | Nest *try* blocks. |
| 108 | 🟢 | Throw an exception. |
| 109 | 🟢 | Use the methods of Throwable. |
| 110 | 🟢 | Create an original exception class. |

**Exploring Database Input/Output**
| 111 | ✅ | Describe a stream. |
| 112 | ✅ | List the main byte and character stream classes. |
| 113 | ✅ | Use predefined streams. |
| 114 | ✅ | Read input from the keyboard. |
| 115 | ✅ | Write output to the monitor. |
| 116 | ✅ | Use the byte streams for file I/O. |
| 117 | ✅ | Read and write binary data. |
| 118 | ✅ | Use the character-based streams for console I/O. |
| 119 | ✅ | Use the character-based streams for file I/O. |
| 120 | ✅ | Create and manipulate streams in Java. |

**Understanding Multithreaded Programming Support**

| 121 | ✅ | Describe the fundamentals of multithreading. |
| 122 | ✅ | Create multiple threads. |
| 123 | ✅ | Describe the life cycle of a thread. |
| 124 | ✅ | Change a thread's priority. |
| 125 | ✅ | Describe the purpose of synchronization. |
| 126 | ✅ | Create synchronized methods. |
| 127 | ✅ | Perform thread communication. |
| 128 | ✅ | Suspend, resume, and stop threads. |

**Creating Applets, Events, and Other Topics**

<p>| 129 | ✅ | Create a Java applet. |
| 130 | ✅ | Explain applet fundamentals, including its architecture. |
| 131 | ✅ | Handle the passing of parameters into an applet from an HTML page. |
| 132 | ✅ | Describe which methods are called by the browser and when they are called. |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>133</td>
<td></td>
<td>Describe procedures for connecting the Java applet to a database.</td>
</tr>
<tr>
<td>134</td>
<td></td>
<td>Create new projects and applications in a development environment.</td>
</tr>
<tr>
<td>135</td>
<td></td>
<td>Design and create a program to meet user needs.</td>
</tr>
</tbody>
</table>

### Using Databases with Java

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>136</td>
<td></td>
<td>Describe Java's relationship to Oracle's database.</td>
</tr>
<tr>
<td>137</td>
<td></td>
<td>List the Java-related driver classes for common databases.</td>
</tr>
<tr>
<td>138</td>
<td></td>
<td>Describe the function of the JDBC package.</td>
</tr>
<tr>
<td>139</td>
<td></td>
<td>Describe the JDBC connection object.</td>
</tr>
<tr>
<td>140</td>
<td></td>
<td>Analyze the process to connect to a database using JDBC package.</td>
</tr>
<tr>
<td>141</td>
<td></td>
<td>Describe the JDBC Statement object.</td>
</tr>
<tr>
<td>142</td>
<td></td>
<td>Describe the use of Java code to iterate through the returned ResultSet.</td>
</tr>
<tr>
<td>143</td>
<td></td>
<td>Review a Java application that will pass variable and user input into a query.</td>
</tr>
<tr>
<td>144</td>
<td></td>
<td>Describe the process to identify and handle Oracle exceptions in Java.</td>
</tr>
<tr>
<td>145</td>
<td></td>
<td>Review a Java application to write data to a database table.</td>
</tr>
</tbody>
</table>

### Preparing for Industry Certification

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>146</td>
<td></td>
<td>Describe the process and requirements for obtaining industry certifications related to the Java Programming (Oracle) course.</td>
</tr>
<tr>
<td>147</td>
<td></td>
<td>Identify testing skills and strategies for a certification examination.</td>
</tr>
<tr>
<td>148</td>
<td></td>
<td>Demonstrate ability to successfully complete selected practice examinations (e.g., practice questions similar to those on certification exams).</td>
</tr>
<tr>
<td>149</td>
<td></td>
<td>Complete an industry certification examination representative of skills learned in this course (e.g., MOS, MTA, IC3).</td>
</tr>
</tbody>
</table>

Legend: ✨Essential ☐Non-essential ☑Omitted

---

**Curriculum Framework**
Comparing Java and PL/SQL Programming

Task Number 39

Compare PL/SQL and Java.

Definition

Comparison should include the following:

- PL/SQL, Java, and SQL are the three key programming languages included in the Oracle database.
- Procedural language/structured query language (PL/SQL) is Oracle's proprietary procedural extension language that included features associated with object orientation for SQL and the Oracle relational database. PL/SQL's general syntax resembles that of Ada or Pascal.
- Java has emerged as the object-oriented programming (OOP) language of choice. Some of the important Java concepts that drove this trend include
  - Java virtual machine (JVM), which provides the fundamental basis for platform independence
  - automated storage management techniques, such as garbage collection
  - language syntax that is similar to that of the C language.

FBLA Competitive Events and Activities Areas

Database Design & Applications

NBEA Achievement Standards for Information Technology

Choose the appropriate language or application development tool for specific tasks.

Task Number 40

Identify the advantages of using PL/SQL in database application development.
Definition

Identification should include the concept that PL/SQL is a portable, high-performance, transaction-processing language that offers the following advantages:

- Tight integration with SQL
- High performance
- High productivity
- Tight security
- Access to predefined packages
- Support for object-oriented programming
- Support for developing web applications and pages

FBLA Competitive Events and Activities Areas

Computer Problem Solving

Database Design & Applications

NBEA Achievement Standards for Information Technology

Choose the appropriate language or application development tool for specific tasks.

Task Number 41

Identify the advantages of using Java in database application development.

Definition

Identification should include the following features of the application:

- Provides a flexible partitioning of Java2 platform, Standard Edition (J2SE) applications for symmetric data access at the JDBC and SQLJ level
- Bridges SQL and the Java2 platform, Enterprise Edition (J2EE) by
  - calling out web components, such as JSP and servlet
  - bridging SQL and web services
  - calling out web services
  - using Oracle JVM as the ERP Integration Hub
  - invalidating the cache.
Learning the Fundamentals of Java as an Object-Oriented Programming Language

Task Number 42

Describe Java's place in computer-language history.

Definition

Description should include

- a description of Java’s creator
- the original intent of the inception of the language
- the three basic principles of Java that make it unique among programming languages: encapsulation, inheritance, and polymorphism.

NBEA Achievement Standards for Information Technology

Evaluate how information technology transforms business processes and relationships.

Identify and evaluate how information technology developments changes the way humans do their work.

Identify and explain the major components of research and development information technologies and their interrelationships.
Use information technology skills in today's learning.

Task Number 43

Explain the fundamentals of object-oriented programming.

Definition

Explaination should include the principles of

- **Inheritance**—It is a form of software reusability in which new classes are created from existing classes by absorbing their attributes and behaviors and adding new capabilities the new classes require. Reusability takes advantage of class relationships where objects of a certain class—such as a class of vehicles—have the same characteristics.

- **Polymorphism**—It is possible to design and implement systems that are more easily extensible. Programs can be written to process generically (as a superclass generically, as subclass objects, or as objects of all existing classes in a hierarchy).

- **Encapsulation**—The code can be protected from outside manipulation by other objects by using the keyword *PRIVATE* or *PROTECTED*. These access modifiers ensure that code will not be altered in an undesirable fashion by outside users.

NBEA Achievement Standards for Information Technology

Identify and explain programming structures.

Task Number 44

Describe the general form of a Java program, including variables, program flow control, main, and methods statements.

Definition

Description of a general form should include reference variables and methods (the two things a class can include) and the three types of methods:

- Main methods—the point of execution
- Constructors—create objects
- Programmer—created methods
Any combination of the three may exist in a class at one time. All program flow control
statements are contained inside a method.

NBEA Achievement Standards for Information Technology

Identify and define object-oriented programming terminology.

---

**Task Number 45**

**Create and compile a block of code.**

**Definition**

Creation of code should include the following syntax:

- the keyword `CLASS`
- the name of the class beginning with a capital letter should follow the keyword `CLASS`
- an opening curly brace
- variable declarations
- the main class definition—including opening and closing curly braces
- an ending curly brace.

Sample code follows:

```java
class SimpleBlock {
    int a = 6;

    public static void main(String args) {
        System.out.println("this is a simple block of code");
        System.out.println("this is the value of a" + a);
    }
}
```

---

NBEA Achievement Standards for Information Technology

Apply design principles to programming tasks.

Demonstrate the ability to code using object-oriented programming.
Select and incorporate appropriate compiler.

Test, debug, and document code.

---

**Task Number 46**

**Identify Java keywords.**

**Definition**

Identification should include Java keywords, such as:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Keyword</th>
<th>Keyword</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>double</td>
<td>int</td>
<td>strictfp**</td>
</tr>
<tr>
<td>Boolean</td>
<td>else</td>
<td>interface</td>
<td>super</td>
</tr>
<tr>
<td>break</td>
<td>extends</td>
<td>long</td>
<td>switch</td>
</tr>
<tr>
<td>byte</td>
<td>final</td>
<td>native</td>
<td>synchronized</td>
</tr>
<tr>
<td>case</td>
<td>finally</td>
<td>new</td>
<td>this</td>
</tr>
<tr>
<td>catch</td>
<td>float</td>
<td>package</td>
<td>throw</td>
</tr>
<tr>
<td>char</td>
<td>for</td>
<td>private</td>
<td>throws</td>
</tr>
<tr>
<td>class</td>
<td>goto *</td>
<td>protected</td>
<td>transient</td>
</tr>
<tr>
<td>const *</td>
<td>if</td>
<td>public</td>
<td>try</td>
</tr>
<tr>
<td>continue</td>
<td>implements</td>
<td>return</td>
<td>void</td>
</tr>
<tr>
<td>default</td>
<td>import</td>
<td>short</td>
<td>volatile</td>
</tr>
<tr>
<td>do</td>
<td>instanceof</td>
<td>static</td>
<td>while</td>
</tr>
</tbody>
</table>

**NBEA Achievement Standards for Information Technology**

Identify and define object-oriented programming terminology.

---

**Task Number 47**

**Describe the standard Java naming conventions.**

**Definition**

Description should include the following:

- All class names are declared with initial letter capitalization (init cap) names.
- Methods and object references are declared in all lowercase names.
• In Java, the second word of a two-word method name is always capitalized, and no spaces are allowed.

NBEA Achievement Standards for Information Technology

Identify and define object-oriented programming terminology.

Task Number 48

Identify the key components of the Software Development Kit (SDK).

Definition

Identification should include

• Java Virtual Machine (JVM) and core Java Application Programming Interface (API) class libraries
• core source code for the Java API libraries
• developer and administrative tools
• demos showing the new SDK 1.2 features
• C header files to extend the core platform with C native methods.

Identification should also include these tools that create and build applications in the foundation of the Java 2 SDK:

• javac—compiler for the Java programming language
• java—launcher for Java technology applications (In this release, a single launcher is used both for development and deployment. The old deployment launcher, jre, is no longer provided.)
• javadoc—API documentation generator
• appletviewer—runs and debugs applets without a web browser.
• jar—manages Java archive (JAR) files
• jdb—Java debugger
• javah—C header and stub generator, used to write native methods
• javap—class file disassembler
• extcheck—utility to detect JAR conflicts

Each of the above are commands that can be executed at the command line.

NBEA Achievement Standards for Information Technology
Differentiate between source and object code.

Identify and define object-oriented programming terminology.

Task Number 49

Describe the JVM.

Definition

Description should include the following:

- JVM is an abstract computer that runs compiled Java programs.
- The JVM is virtual because it is generally implemented in software on top of a real hardware platform and operating system.
- All Java programs are compiled for the JVM; therefore, the JVM must be implemented on a particular platform before compiled Java programs can run on that platform.

NBEA Achievement Standards for Information Technology

Differentiate between source and object code.

Identify and define object-oriented programming terminology.

Distinguishing Key Elements of Java

Task Number 50

Use Java's primitive data types.

Definition

Use of simple data types should involve the creation of a program that declares a variable of type int, byte, short, char, long, double, float, or Boolean. Description of each variable follows:
<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Description</strong></th>
<th><strong>Size/Format</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(integers)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>Byte-length integer</td>
<td>8-bit two's complement</td>
</tr>
<tr>
<td>short</td>
<td>Short integer</td>
<td>16-bit two's complement</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>32-bit two's complement</td>
</tr>
<tr>
<td>long</td>
<td>Long integer</td>
<td>64-bit two's complement</td>
</tr>
<tr>
<td><strong>(real numbers)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>Single-precision floating point</td>
<td>32-bit IEEE 754</td>
</tr>
<tr>
<td>double</td>
<td>Double-precision floating point</td>
<td>64-bit IEEE 754</td>
</tr>
<tr>
<td><strong>(other types)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>A single character</td>
<td>16-bit Unicode character</td>
</tr>
<tr>
<td>Boolean</td>
<td>A Boolean value (true or false)</td>
<td>true or false</td>
</tr>
</tbody>
</table>

NBEA Achievement Standards for Information Technology

Demonstrate the ability to code using object-oriented programming.

---

**Task Number 51**

**Initialize and assign variables.**

**Definition**

Initialization of variables should include specifying values for simple data types, which should be in the form of assigning a value of the appropriate data type to the declared primitive (e.g., `int a = 5;`).

**Task Number 52**

**Describe the scope rules of a variable.**

**Definition**

Description should include

- using curly braces "{}" that begin and end the operations of a method (Any object or variable declared within the scope or curly braces of a method is only available to the code within the boundaries of the curly braces.)
• declaring objects or variables at the beginning of the class definition that need to be accessed by the rest of the class.

NBEA Achievement Standards for Information Technology

Demonstrate the ability to code using object-oriented programming.

Task Number 53

Apply type conversion (casting) in expressions.

Definition

Application of type conversion should include

• changing from one data type to another
• casting by putting the result data type in parentheses in front of the variable
• adding implicit casts that Java performs if there will be no loss of precision.

Task Number 54

Use the arithmetic operators.

Definition

Using arithmetic operators should include outlining the available operators in Java and their possible uses:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>op1 + op2</td>
<td>Adds op1 and op2</td>
</tr>
<tr>
<td>-</td>
<td>op1 - op2</td>
<td>Subtracts op2 from op1</td>
</tr>
<tr>
<td>*</td>
<td>op1 * op2</td>
<td>Multiplies op1 by op2</td>
</tr>
<tr>
<td>/</td>
<td>op1 / op2</td>
<td>Divides op1 by op2</td>
</tr>
<tr>
<td>%</td>
<td>op1 % op2</td>
<td>Computes the remainder of dividing op1 by op2</td>
</tr>
</tbody>
</table>

These short-cut operators increment or decrement a number by one:
<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>op++</td>
<td>Increments op by 1; evaluates to the value of op before it was incremented</td>
</tr>
<tr>
<td>++</td>
<td>++op</td>
<td>Increments op by 1; evaluates to the value of op after it was incremented</td>
</tr>
<tr>
<td>--</td>
<td>op--</td>
<td>Decrements op by 1; evaluates to the value of op before it was decremented</td>
</tr>
<tr>
<td>--</td>
<td>--op</td>
<td>Decrements op by 1; evaluates to the value of op after it was decremented</td>
</tr>
</tbody>
</table>

These are the Java programming language's other arithmetic operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+op</td>
<td>Promotes op to int if it is a byte, short, or char</td>
</tr>
<tr>
<td>-</td>
<td>-op</td>
<td>Arithmetically negates op</td>
</tr>
</tbody>
</table>

NBEA Achievement Standards for Information Technology

Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).

---

**Task Number 55**

**Compare relational and logical operators.**

**Definition**

Comparison should include a summary of relational and conditional operators, shift and logical operators, and other operators, which the following tables illustrate.

**Relational Operators** (determine the relationship between two values):

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Returns true if</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>op1&gt;op2</td>
<td>op1 is greater than op2</td>
</tr>
<tr>
<td>&gt;=</td>
<td>op1 &gt;= op2</td>
<td>op1 is greater than or equal to op2</td>
</tr>
<tr>
<td>&lt;</td>
<td>op1 &lt; op2</td>
<td>op1 is less than op2</td>
</tr>
<tr>
<td>&lt;=</td>
<td>op1 &lt;= op2</td>
<td>op1 is less than or equal to op2</td>
</tr>
<tr>
<td>==</td>
<td>op1 == op2</td>
<td>op1 and op2 are equal</td>
</tr>
<tr>
<td>!=</td>
<td>op1 != op2</td>
<td>op1 and op2 are not equal</td>
</tr>
</tbody>
</table>
**Conditional Operators** (form multi-part decisions):

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Returns true if</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>op1 &amp;&amp; op2</td>
<td>op1 and op2 are both true, conditionally evaluates op2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>! op</td>
<td>op is false</td>
</tr>
<tr>
<td>&amp;</td>
<td>op1 &amp; op2</td>
<td>op1 and op2 are both true, always evaluates op1 and op2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>op1</td>
</tr>
<tr>
<td>^</td>
<td>op1 ^ op2</td>
<td>if op1 and op2 are different—that is if one or the other of the operands is true but not both</td>
</tr>
</tbody>
</table>

**Shift and Logical Operators** (shifts the bits of the left-hand operand over by the number of positions indicated by the right-hand operand, thereby performing logical functions on their operands):

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;</td>
<td>op1 &gt;&gt; op2</td>
<td>shift bits of op1 right by distance op2</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>op1 &lt;&lt; op2</td>
<td>shift bits of op1 left by distance op2</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td>op1 &gt;&gt;&gt; op2</td>
<td>shift bits of op1 right by distance op2 (unsigned)</td>
</tr>
</tbody>
</table>

**Other Operators:**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?:</td>
<td>op1 ? op2 : op3</td>
<td>If op1 is true, returns op2. Otherwise, returns op3.</td>
</tr>
<tr>
<td>[]</td>
<td>type []</td>
<td>Declares an array of unknown length, which contains type elements.</td>
</tr>
<tr>
<td>[]</td>
<td>type[ op1 ]</td>
<td>Creates an array with op1 elements. Must be used with the new operator.</td>
</tr>
<tr>
<td>[]</td>
<td>op1[ op2 ]</td>
<td>Accesses the element at op2 index within the array op1. Indices begin at zero and extend through the length of the array minus one.</td>
</tr>
<tr>
<td>.</td>
<td>op1.op2</td>
<td>Is a reference to the op2 member of op1.</td>
</tr>
</tbody>
</table>
(*) op1(params)

Declares or calls the method named op1 with the specified parameters. The list of parameters can be an empty list. The list is comma-separated.

(type) (type) op1

Casts (converts) op1 to type. An exception will be thrown if the type of op1 is incompatible with type.

new new op1

Creates a new object or array. op1 is either a call to a constructor or an array specification.

instanceof op1 instanceof op2

Returns true if op1 is an instance of op2

& op1 & op2

bitwise and

| op1 | op2

bitwise or

^ op1 ^ op2

bitwise xor

~ ~op2

bitwise complement

---

**Task Number 56**

**Input characters from the keyboard.**

**Definition**

Inputting characters from the keyboard in applications should include code, such as the code in the following program that defines a method called minMax. MinMax takes two integers and creates an array that is populated with all integers between the two integers.

```java
import java.io.*;
public class KeyBoardInput {
    public int[] minMax(int lower, int upper) {
        int[] arr = new int[(upper-lower) + 1];
        for (int i=0; i<arr.length; i++)
            arr[i] = lower++;
        return arr;
    }
    public static void main(String args[])
        throws Exception {
            InputStreamReader reader = new InputStreamReader(System.in);
        }
    InputStreamReader reader = new InputStreamReader(System.in);
    ```
22 BufferedReader input=new BufferedReader(reader);
23 System.out.println("Please enter a number");
24 String letters= input.readLine();
25 int numbers =Integer.parseInt(letters);
26 System.out.println("Please enter the upper number");
27 String moreInput =input.readLine();
28 int moreNumbers=Integer.parseInt(moreInput);
29 int theArray[];
30 KeyboardInput theRange=new KeyboardInput();
31 theArray = theRange.minMax(numbers, moreNumbers);
32 System.out.print("The array:[");
33 for(int i=0;i<theArray.length;i++)
34 { System.out.print(theArray[i]+" ");
35 } System.out.println(" ");

Related Standards of Learning

Mathematics

COM.13
The student will implement various mechanisms for performing iteration with an algorithm

Task Number 57

Navigate the Java API.

Definition

Navigation of the Java API includes moving through a library of pre-written code or classes that are organized in a hierarchical structure.
Task Number 58

Build and execute Java applications, using a development tool.

Definition

Building Java applications with a development tool in an integrated development environment or a basic editor should be executable to the instructor's requirements.

NBEA Achievement Standards for Information Technology

Choose the appropriate language or application development tool for specific tasks.

Maintain and reengineer existing code.

Test, debug, and document code.

Task Number 59

Modify Java application source code.

Definition

Modification of Java application source code should involve an iterative process whereby previously written code is continually improved and upgraded as new skills are learned.

NBEA Achievement Standards for Information Technology

Maintain and reengineer existing code.

Controlling Program Flow
Task Number 60
Identify situations in which to use control statements.

Definition

Identification should include

- diagramming the flow of the program
- selecting the control structure that achieves the desired results.

Other control structures include switch, case, and exception handling.

NBEA Achievement Standards for Information Technology

Identify and define the coding task.

Task Number 61
Use *if-then* and *if-then-else* statements.

Definition

Use of the *if-then* statement, the most basic form of program flow control, should include

- applying the statement to a program to execute a defined section of code only if a test condition results to true
  - Example: If there is candy in the candy jar, distribute the candy to a student and decrease the candy count.

```c
void distributeCandy() {
    // the "if" clause: must have candy
    if (isCandy)
    {
        // the "then" clause: distribute candy to students
        numberCandy--;
    }
}
```

If this test evaluates to false (there is no candy), control jumps to the end of the *if-then* statement.

The use of the *if-then-else* statement includes: The *if-then-else* statement
• providing a secondary or else path of execution when an if clause evaluates to false.
  o Example: Use an if-then-else statement in the distributeCandy method to take some when there is no candy.

```java
double distributeCandy() {
  // the "if" clause: must have candy
  if (isCandy)
  {
    // the "then" clause: distribute candy to students
    numberCandy--;  
  } else
  {
    System.err.println("The candy jar is empty");
  }
}
```

NBEA Achievement Standards for Information Technology

Demonstrate the ability to code using object-oriented programming.

Develop both procedural and object-oriented programs.

-----------------------------

**Task Number 62**

**Use switch statements.**

**Definition**

A switch statement can be used with byte, short, char, and int primitive data types, along with enumerated types and the string class.

Sample code follows (from oracle.com):

```java
public class SwitchDemo {
  public static void main(String[] args) {
    int month = 8;
    String monthString;
    switch (month) {
    case 1: monthString = "January";  
      break;
    case 2: monthString = "February";
      break;
    case 3: monthString = "March";
      break;
    default: break;
    }
  }
}
```
break;
case 4: monthString = "April";
break;
case 5: monthString = "May";
break;
case 6: monthString = "June";
break;
case 7: monthString = "July";
break;
case 8: monthString = "August";
break;
case 9: monthString = "September";
break;
case 10: monthString = "October";
break;
case 11: monthString = "November";
break;
case 12: monthString = "December";
break;
default: monthString = "Invalid month";
break;
}
System.out.println(monthString);
}

Note: In the code above, August is printed to standard output. The body of a switch statement is known as a switch block. A statement in the switch block can be labeled with one or more case or default labels. The switch statement evaluates its expression, then executes all statements that follow the matching case label. Switch statements are used for conditional processing, providing a number of possible execution paths.

**NBEA Achievement Standards for Information Technology**

Demonstrate the ability to code using object-oriented programming.

Develop both procedural and object-oriented programs.

---

**Task Number 63**

**Use for loop statements.**
Definition

The use of the full form of the for loop statement is detailed as follows:

\[ \text{for (int } i=0; \text{ i<arr.length; i++)} \]
\[ \{ \]
\[ \text{arr}[i] = \text{lower++;} \]
\[ \} \]
\[ \text{return arr;} \]

\[ \text{public static void main (String args[])} \text{throws Exception} \]
\[ \{ \]
\[ \text{InputStreamReader reader } = \text{new InputStreamReader(System.in);} \]
\[ \}
\[ \text{BufferedReader input } = \text{new BufferedReader(reader);} \]
\[ \}
\[ \text{System.out.println("Please enter a number");} \]
\[ \}
\[ \text{String letters } = \text{input.readLine();} \]
\[ \}
\[ \text{int numbers } = \text{Integer.parseInt(letters);} \]
\[ \}
\[ \text{System.out.println("Please enter the upper number");} \]
\[ \}
\[ \text{String moreInput } = \text{input.readLine();} \]
\[ \}
\[ \text{int moreNumbers } = \text{Integer.parseInt(moreInput);} \]
\[ \}
\[ \text{int theArray[ ];} \]
\[ \}
\[ \text{KeyBoardInput theRange } = \text{new KeyBoardInput();} \]
\[ \}
\[ \text{theArray } = \text{theRange.minMax(numbers, moreNumbers);} \]
\[ \}
\[ \text{System.out.print("The array:");} \]
\[ \}
\for (int i=0; i<theArray.length; i++)

Related Standards of Learning

Mathematics

COM.13
The student will implement various mechanisms for performing iteration with an algorithm

NBEA Achievement Standards for Information Technology
Demonstrate the ability to code using object-oriented programming.

Develop both procedural and object-oriented programs.

---

Task Number 64

Use the **while** loop.

**Definition**

Using the **while** loop should include a statement that will continue executing a block of statements for the duration a Boolean expression remains true. The expression is evaluated at the top of the loop. Sample code follows:

```java
while (Boolean expression) {
    statement(s)
}
```

Using the **do-while** statement should loop over a block of statements while a Boolean expression remains true. The expression is evaluated at the bottom of the loop, so the statements within the "do-while" block execute at least once. Sample code follows:

```java
do {
    statement(s)
} while (expression);
```

---

NBEA Achievement Standards for Information Technology

Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).

---

Task Number 65

Use the **do-while** loop.

**Definition**

Using the **do-while** loop should execute a condition one time before the condition of the loop is checked.
Task Number 66

Use break and continue statements effectively.

Definition

Using break and continue statements should include transferring control out of one structure and into another. For example:

```java
// Read input until a q is received.
class Break2 {
    public static void main(String args[]) throws java.io.IOException {
        char ch;
        for( ; ; ) {
            ch = (char) System.in.read(); // get a char
            if(ch == 'q') break;
        }
        System.out.println("You pressed q!");
    }
}
```

Employment of break and continue statements can also be with inner loops:

```java
// Using break with nested loops.
class Break3 {
    public static void main(String args[]) {
        for(int i=0; i<3; i++) {
            System.out.println("Outer loop count: "+ i);
            System.out.print(" Inner loop count: ");
            int t = 0;
            while(t < 100) {
                if(t == 10) break; // terminate loop if t is 10
                System.out.print(t + " ");
                t++;
            }
        }
        System.out.println();
    }
}
```
System.out.println("Loops complete.");

NBEA Achievement Standards for Information Technology

Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).

Introducing Classes, Objects, and Methods

Task Number 67

Identify class and method definitions.

Definition

Identification of a class and method definition must adhere to the following specific syntax format:

Class definition syntax

Access Modifier(optional) Class_name {
    variable declaration
    object creation
    method definitions
    constructors
    main methods
    programmer defined methods

}

Method definition syntax

access modifier return type classname (data type param_name) {

}

NBEA Achievement Standards for Information Technology
Apply design principles to programming tasks.

Task Number 68

Differentiate between classes and objects.

Definition

Differentiation should include examining the code that defines the parameters of a class, but it does not have a space in memory, does not know anything about itself, and does not know how to perform any actions. Objects, on the other hand, know things about themselves (fields or variables) and know how to perform actions.

NBEA Achievement Standards for Information Technology

Task Number 69

Describe an object and its operations (or methods).

Definition

Description should include the following:

- Objects are defined by their class files to know things about themselves, such as how to perform when asked.
- An object must be created before the fields are available or before it can perform any actions.

Task Number 70

Create a method, and return a value from a method.

Definition

Creation of a method, which defines a starting point or runnable node, should occur within a class. Each method should perform one task or several closely related tasks. Access by other objects is determined by the modifier given to the method.
Returning a value from a method requires two things: the data type of the returned value in the method definition and a return statement in the body of the method. An example of a method with a return statement is as follows:

```java
1 class Person {
2
3  public String fullName;
4  private int age;
5  public char gender;
6  public static int retirementAge = 65;
7  // This is a no argument constructor
8
9  public Person()
10  {
11    fullName = "John Doe";
12    age = 0;
13    gender = '?';
14  }
15
16  // This constructor receives an String to initialize the name
17  public Person(String aNewName)
18  {
19    this(); // We are calling the first constructor
20    /* From Developer.com The keyword "this" has three basic uses:
21       1. To bypass local variables or parameters that hide member variables having the same
22          name, in order to access
23          the member variable.
24       2. To make it possible for one overloaded constructor to invoke another overloaded
25          constructor in the same class.
26       3. To pass a reference to the current object to a method belonging to a different object (as
27          in implementing
28          callbacks, for example).
29     */
30    fullName = aNewName;
31    System.out.println(this);
32  }
33
34  public void printName()
35  {
36    System.out.println(fullName);
37  }
38
39  public void setFullName(String newName)
40  {
41    fullName = newName;
42  }
43
44  public String getFullName()
45  {
46    return fullName;
47  }
48  }
```
public int getAge()
    { return age; }

public void setAge(int aNewAge)
    { age = aNewAge; }

public static int getRetirementAge()
    { return retirementAge - age; }

Task Number 71

Use parameters in a method.

Definition

Using parameters in a method should include

- defining the required data type for the method to execute
- naming the past objects or primitives. For example:

```java
public void setAge(int aNewAge)
    { age = aNewAge; }
```

The data type int (represented above) is found in the parentheses of the method signature. The temporary name for the parameter is aNewAge.

Task Number 72

Add a constructor to a class.

Definition

Adding a construction necessitates use of a default, no-argument constructor, supplied by Java for every class. If an overloaded constructor is written, the default, no-argument constructor must be hard-coded into the method.
In the code above, a constructor is created that sets the value of name, age, and gender for a person.

NBEA Achievement Standards for Information Technology

Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).

Task Number 73

Overload a constructor.

Definition

Overloading, for the purpose of customizing an object at the time of creation, should be performed in the same manner as overloading a regular method:

- Java supplies a default, no-argument constructor, even if one is not created.
- If an overloaded constructor is written, the default no-argument constructor is no longer available unless hard-coded into the class definition.

Task Number 74

Describe garbage collection and the finalize () method.

Definition

Description should include the following:

- Whenever there is a need for memory and an object is no longer referenced, garbage collection is performed automatically by the Java Virtual Machine.
- It may be desirable to run the garbage collection explicitly by calling the gc method in the System class.
Before an object is garbage-collected, the garbage collector gives the object an opportunity to clean up after itself through a call to the object's finalize method, a process known as finalization.

NBEA Achievement Standards for Information Technology

Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).

---

Task Number 75

Apply the **this** reference.

**Definition**

Applying the **this** reference should include using the keyword to refer to the current object in code, such as in the following, where the keyword **this** refers to the `Dog` object being created:

```java
public class Dog {
    private int numLegs, numEyes, numTeeth;
    public Dog (int numLegs, int numEyes, int numTeeth) {
        this.numLegs = numLegs;
        this.numEyes = numEyes;
        this.numTeeth = numTeeth;
    }
}
```

---

Task Number 76

**Explain the purpose and importance of the Object class.**

**Definition**

Explanation should include that the Object class

- defines and implements behavior that every class in the Java system needs
- is the most general of all classes
- has immediate subclasses and other classes near the top of the hierarchy that implement general behavior
- has classes near the bottom of the hierarchy that provide for more specialized behavior.
Using Data Types and Operators

Task Number 77

Use single and multidimensional arrays.

Definition

Using arrays should include declaring them first, such as by the following:

```java
float[] x; x = new float[8];
boolean[] flags = new boolean[1024];
String[] names = new String[32];
Point[] ideal = new Point[1000];
int c[] = new int[12]
double[] array1, array2;
double[] array1 = new double[10];
```

Java supports declarations of arrays with brackets either after the data type of the array or after the name of the array.

NBEA Achievement Standards for Information Technology

Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).

Task Number 78

Use various formats to initialize arrays.

Definition

Initialization should include designating each element separately, through a set, or within a loop. Sample code of an array declaration follows:

```java
int[][] myArray = new int[3]
```

NBEA Achievement Standards for Information Technology
Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).

Task Number 79

Apply the bitwise operators.

Definition

Application of bitwise operators necessitates the conversion of base-10 numbers to base-2, or binary numbers, or conversion of base-10 to base-16 numbers.

Application of bitwise logical operators should include

- AND (&)
- inclusive OR (|)
- exclusive OR (^).

Bitwise shift operators should include

- left shift (<<)
- right shift (>>)
- compliment (~).

NBEA Achievement Standards for Information Technology

Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).

Task Number 80

Create objects of type String and utilize their methods.

Definition

Creating objects of type String can be accomplished in one of two ways:

1. String name = new String (jordan)
2. String name = "jordan"
Utilizing the methods defined by the String class consists of writing dot notation in the form of object name.methodL). For example, utilizing the String created above, a method called \textit{jordan} to find the third character in the string would be written as follows:

\texttt{jordan.charAt(3);}

This would return an \textit{r} to the calling program.

\textbf{NBEA Achievement Standards for Information Technology}

\textbf{Code common tasks (e.g., creating, adding, deleting, sorting, and updating records).}

\section*{Understanding Methods and Classes}

\section*{Task Number 81}

\textbf{Define the public- and private-access modifiers.}

\textbf{Definition}

Definitions should include

- \textit{Public access modifier}—A variable or class declared \textit{public} is available to any other object, including subclasses and classes in other packages.
- \textit{Private access modifier}—Variables, methods, or constructors declared \textit{private} can only be accessed within the declared class itself.

\section*{Task Number 82}

\textbf{Pass and return objects/primitive parameters to methods.}

\textbf{Definition}

Passing objects or primitives involves the recognition of the method signature that is located immediately after the method name in parentheses. In the parentheses, the type of primitive/object required by the code to execute is followed by a name given to the primitive or
object. In order to pass a parameter, the appropriate object or primitive must be placed in the parenthesis when the method is called.

The general format for the calling of an object is as follows:

```
objectName.methodName(required object/primitive);
```

Returning an object or primitive from a method requires that the method be defined with the keyword `return`. Once the method is called, the returned object must be assigned to a space in memory either declared at the time the method was called or previously declared in the program. When a method returns a value, it must return it into a container of the appropriate type. If a method returns an integer, there must be a variable of type integer waiting to receive it. Sample code follows:

```
String number = new String("2");
int a;
a = Integer.parseInt(number);
```

In this example, the variable `number` is the required parameter as defined in the API. The `Integer.parseInt` method's signature asks for a string and returns an `int` representation of that string to the assignment operator.

**Task Number 83**

**Use class variables and class methods.**

**Definition**

Using class variables and methods, otherwise referred to as static variables and methods, should include

- initializing them only once, before initializing instance variables
- accessing the variable directly by the class name.

Note: Static variables belong to the class, not to the instance or object.

**Task Number 84**

**Apply recursive methods.**

**Definition**
Applying a recursive method should include

- making the method call itself until an end point is reached
- using these methods in the construction of a binary tree that executes a sort.

**Task Number 85**

Create and use recursive methods and variables.

**Definition**

Creation of a recursive method should include defining a method and then calling the method within itself. For example:

```java
public static int powersOfThree(int n){
    if (n == 0){
        return 1;
    }
    else {
        return 3* powersOfThree(n-1);
    }
}
```

**Task Number 86**

Describe nested and inner classes.

**Definition**

Description of a nested or inner class should include the following:

- They are also referred to as anonymous inner classes because they can be declared within the body of an outer class and have no name.
- Such classes are used primarily for event handling.
- An inner class object is allowed to directly access all the variables and methods of the outer class object that defined it.
- An inner class defined in a method is allowed to access directly all the instance variables and methods of the outer class object that defined it and any final local variables in the method.

**Using Inheritance**
Task Number 87

Use inheritance to define new classes.

Definition

Using inheritance requires the keyword *extends* and, because a subclass receives all of the public and protected fields and methods from the super class, it helps avoid rewriting code.

NBEA Achievement Standards for Information Technology

Maintain and reengineer existing code.

Task Number 88

Describe the difference between a superclass and a subclass.

Definition

Description of the difference between the superclass and subclass should include the following:

- The subclass is a more specific variation of the more general superclass.
- A superclass may be declared abstract to force inheritance, since it could be too general to be actually created.
- Each object instantiated from a subclass is a member of the superclass, but each object instantiated from the superclass is not implicitly a member of the subclass (because the subclass has fields and methods that the superclass does not possess).

Task Number 89

Explain how inheritance affects member access.

Definition

Explanation should include the discussion of the following rule: A subclass inherits all of the members in its superclass that are accessible to that subclass unless the subclass explicitly hides a member variable or overrides a method. (Note that constructors are not members and are not inherited by subclasses.)
Task Number 90

Use *super* to call superclass constructors and members.

Definition

Usage of the keyword *super* allows the constructor to utilize the prewritten code from a parent class. In this way, the parent constructor is allowed to do its job and then the child constructor can add additional functionality to it. *Super* must be the first line in the constructor, if it is used. The following class is a child class of a class called *SuperClassName*. Coding for the constructor follows:

```java
class LittleClass extends BigClass{
    int a, b, c;
    LittleClass(){
        a = 6;
        b = 7;
        c = 8;
    }
    //body of class
}//end class definition
```

Usage should include the uses of the *super* keyword:

- If a class overrides a method in a superclass, the *super* keyword can be used to bypass the overridden version in the class and access the version in the superclass.
- If a local variable in a method or a member variable in a class hides a member variable in the superclass (with the same name), the *super* keyword can be used to access the member variable in the superclass.
- *Super* can also be used in a constructor of the class to invoke an overloaded constructor in the superclass.

Sample code for accessing superclass members follows:

```java
class BigSuperClass{
    int info;
    /*explicit coding of the default no argument constructor is required if another overridden
    constructor is created */
    public BigSuperClass(){
    }
```
/* An overloaded constructor requiring an integer parameter. Note that the signature is changed in order to cause the overloading. */
public BigSuperClass(int val){
    System.out.println("Inside the BigSuperClass constructor. ");
    System.out.println("Setting BigSuperClass instance " + "variable info to " + val);
    info = val;
    System.out.println();//blank line
}
//end BigSuperClass constructor
}
//end BigSuperClass class definition
//=----------------------------------------------------------------------------------=
class SuperThree extends BigSuperClass{
/* Instance variable in subclass has same name as instance variable in BigSuperClass */
int info;
//Subclass no arg constructor
public SuperThree(){
    Duty/Concept Area: Using Inheritance
    91
    //call the constructor requiring parameters from BigSuperClass
    super(700);
    System.out.println("In subclass constructor.");
    System.out.println("Setting subclass instance variable info " + "to 56");
    info = 56;
    System.out.println();//blank line
} //end subclass constructor
//=----------------------------------------------------------------------------------=

/* The following method illustrates use of this and super and clarifies the identification of the local variable called info, instance variable of subclass called info, and instance variable of BigSuperClass called info. All three variables have the same name. */
void confusingVariableNames(){
    int info = 75
    //local variable
    System.out.println("In method confusingVariableNames");
System.out.println("Local variable info = "+ info);
System.out.println(  
"Subclass instance variable info = "+ this.info);
System.out.println(  
"BigSuperClass instance variable info = "+ super.info);
}
//end method confusingVariableNames
//-------------------------------//
public static void main(String[] args){
SuperThree objOfSubClass = new SuperThree();
System.out.println("Executing the main method");
System.out.println(  
"Subclass instance variable info = "+ objOfSubClass.info);
System.out.println(); //blank line
objOfSubClass.confusingVariableNames();
}
//end main method
}
//End SuperThree class definition.

Task Number 91

Create a multilevel class hierarchy.

Definition

Creation of a multilevel class hierarchy should include a superclass extended, and the extended class should then be extended by another class. For example, to extend the class SuperThree, the following code would be necessary:

    public class SuperFour extends SuperThree{
    public String aboutMe = "this class is a part of a multilevel hierarchy";
    public SuperFour() {
    super();
    
    } /*The end of the SuperFour constructor which calls upon the superThree constructor, which calls on the BigSuperClass constructor */
    public void printAboutMe() {

System.out.println("Inside the printAboutMe method, the aboutMe variable is :" + aboutMe);
System.out.println("From the SuperThree Class, the value of the variable info is ":" + info);
/* end the method that prints the value of the aboutMe variable*/
//begin the main method, which is the point of execution for the SuperFour class
public static void main(String args[]) {
    SuperFour secondGeneration = new SuperFour();
    secondGeneration.printAboutMe();
}

Task Number 92

Explain cases when constructors are called in a class hierarchy.

Definition

Explanation should include the following:

- Java decides which constructor is to be used based on the parameters that are supplied at
  the time it is called.
- If no parameters are supplied, then the default, no-argument constructor is used.
- If an integer is supplied in the calling code, the second constructor is used.

The code below would cause the second constructor to be called to create an object of type
Stacking:

class DoSomeStacking{
    public static void main(String args[]) {
        Stacking myStack = new Stacking(3);
        //Java selects the constructor that requires an integer to execute
        //and creates the object with an array called items that can hold 3 elements
    }
}

Task Number 93

Apply superclass references to subclass objects.

Definition

Application should include Java member variables that can access a hidden member variable
through the superclass. Sample code of a superclass and subclass pair follows:
class SuperDouper {
    Number aNumber;
}
class SubbieDubbie extends SuperDouper {
    Float aNumber;
}

Task Number 94

Demonstrate how to override methods.

Definition

Demonstration should include writing code that accomplishes the override. The code below shows the method `printAboutMe` will be overridden by retaining the identical method declaration:

class SuperFive extends SuperFour {
    /* Just one method in the SuperFive class and its job is to override the printAboutMe() method from the super class*/
    public void printAboutMe() {
        System.out.println("Let’s not even print the value of a variable here, just say Hi!");
    }
    public static void main(String args[]) {
        SuperFive objOfSuperFive = new SuperFive();
        objOfSuperFive.printAboutMe();
    }
}

Task Number 95

Create abstract methods and classes.

Definition

Creation of an abstract method should include the modifier `abstract` and adhere to the following:

- An abstract class may contain abstract methods or concrete methods, or a combination of the two.
- An abstract class cannot be instantiated.
- An abstract method is similar to an interface, but not identical; for instance:
  - An abstract method is utilized through the inheritance from a parent class.
  - An interface is implemented from outside of the hierarchical lineage.
The following is an example of the creation of an abstract method:

```java
public abstract class GenericVehicle{

    /*Creation of a class that is too general to really be useful, but that can act as the "organizational tool" for subclasses. The abstract modifier ensures that the class can never be instantiated, but it can be extended. If it is extended, all methods must be overridden.*/

    int noOfDoors;
    int noOfWheels;
    int tireSize;

    public abstract void go( );
    public abstract void stop( );

}
```

**Task Number 96**

**Use the keyword `final` to prevent inheritance.**

**Definition**

Using the `final` modifier should include applying it in situations where the programmer does not want the class or field changed by inheritance or reassignment. A totally abstract class may contain only final variables and abstract methods as follows:

```java
public abstract class GenericVehicle{

    /*Creation of a class that is too general to be really useful can act as the "organizational tool" for subclasses. The abstract modifier ensures that the class can never be instantiated, but it can be extended. If it is extended, all methods must be overridden.*/

    final int noOfDoors = 2;
    final int noOfWheels = 4;
    final int tireSize = 16;

    //All subclasses of GenericVehicle will have 2 doors, 4 wheels, and size 16 tires, but each subclass of GenericVehicle will be required to supply its own code for stop() and go();

    public abstract void go( ) ;
    public abstract void stop( ) ;

    /*include one more method to show that not all methods in an abstract class must be abstract as well */
```
Using Packages and Interfaces

Task Number 97

Create and implement an interface, utilizing its variables.

Definition

Creation of an interface should include

- declaring a class as an interface
- allowing no more than two kinds of members in the interface definition:
  - Methods, which are implicitly abstract
  - Variables, which are implicitly constant (final)

An example of an interface follows:

```java
interface ImplementMe{
    final int noLines = 10;
    public void showData();
}
```

//===================================//
/*Show that interfaces can be inherited as well. The extended subclass is also an interface and is not bound by the rule that says the implementation must be supplied if an interface is implemented.*/

```java
interface ImplementMeToo extends ImplementMe{
    public void showMoreData();
}
```

//ImplementMeToo also has a method called showData from its superclass, and both must be included in a program that implements the interface ImplementMeToo//
Using an interface reference and variables should be done within the context of a program which implements the interface, extends another class, and utilizes the value of one of the variables. An example follows:

class Demonstration extends Object{
    //override the toString method that is inherited from the class Object
    public String toString() {
        return "our own version of toString";
    }
    //create a new method that is not inherited from the class object
    public String getAnotherString() {
        return "this is from the method getAnotherString in Demonstration";
    }
}

//The following class implements, inherits and prints the value of the final variable
//from the interface.
class AnotherDemonstration extends Demonstration implements ImplementMeToo{
    public void showData() {
        System.out.println("I don't have any data to show!");
    }
    public void showMoreData() {
        //Nothing even has to be here, as long as the curly braces are supplied
    }
    public void printTheFinal() {
        System.out.println("The value of noLines is "+ noLines);
    }
}

Task Number 98

Extend an interface.

Definition

Extension should be written to anticipate multiple uses for the interface and allow users to upgrade. Sample code for an extension follows:

//create an interface
public interface StockWatcher {
    final String
    sunTicker = "SUNW";
    final String oracleTicker = "ORCL";
    final String ciscoTicker = "CSCO";
    void valueChanged(String tickerSymbol, double newValue);
}
Task Number 99

Create and import a package.

Definition

Creating a package would consist of a package statement as follows:

```java
package software;
```

placed at the beginning of every file to be included in that package.

Importing a package of programs to be used in the code in a class requires an import statement such as:

```java
import software.*;
```

The "*" character provides direction to import all classes in that package.

Task Number 100

Describe how a package affects access to its programs.

Definition

Description should include the following:

- Members of a package receive protected access by default.
- Only members of a package declared public can be accessed by outside objects.

Task Number 101

Explain the purpose of the Thread class and the Runnable interface.

Definition

Explanation should include that Thread class involves multitasking and that there are two ways that an object can have a thread attached to it:

- The object must extend the class Thread
- The object must implement the Runnable interface

The Runnable interface causes the run() method to be implemented and the run() initializes the thread.
Handling Exceptions

Task Number 102

Explain the purpose of exception handling.

Definition

Explanation should include

- a definition of *exception handling*
- examples of indefinite program execution or problematic program termination
- the way these problems are anticipated and dealt with by an adept programmer
- the following exception hierarchy:

Object

- Throwable
  - Exception
  - IOException
  - ArithmeticException
  - ArrayStoreException
  - IllegalArgumentException
  - IndexOutOfBoundsException
  - NullPointerException
  - RuntimeException
  - FileNotFoundException
  - EndOfFileException
- Error

Object

Throwable

Exception

IOException

ArithmeticException

ArrayStoreException

IllegalArgumentException
IndexOutOfBoundsException
NullPointerException

RuntimeException

FileNotFoundException
EndOfFileException

Error

**Task Number 103**

**Write code to demonstrate how try and catch work together to handle an exception.**

**Definition**

Creation of code for an exception should include the following components:

- The program should *try* to execute the statements contained within a block of code. (A *try* block of code is a group of one or more statements surrounded by braces after the keyword *try.*)
- If an exceptional condition within that block is detected, the exception object of a specific type is thrown out. When an exceptional condition occurs within a method, the method may instantiate an exception object and hand it off to the runtime system. This is accomplished using the *throw* keyword. To be useful, the exception object should contain information about the exception, including its type and the state of the program when the exception occurred.
- *Catch* should be used and the exception object processed, using code. The runtime system begins its search with the method in which the exception occurred and searches backward through the call stack until it finds a method that contains an appropriate exception handler (catch block).
- An exception handler is appropriate if the type of the exception thrown is the same as the type of exception handled by the handler, or is a subclass of the type of exception handled by the handler.
- Optionally a block of code should be executed, designated by the *finally* keyword, which needs to be executed regardless of the occurrence of an exception. (Code in the *finally* block is normally used to perform some type of cleanup or closure of an opened file.)

A simple program that requires exception handling follows:

```java
import java.io.*;
public class HelloName{
    public static void main(String args[]) {
        final int LENGTH = 255;
```
byte buffer[] = new byte [LENGTH]
System.out.print("Enter your name: ");
try{
    System.in.read(buffer, 0, LENGTH);
}
catch (Exception e) { }
String name = new String(buffer);
System.out.println("Hello, " + name.trim() + "!");
}

Task Number 104

Create an exception handler.

Definition

Creation of an exception handler should replicate the structure of the following example:

import java.io.*;
public class MakeSomeErrors {

    //a method to read in keystrokes from the command (as a String) and
    //convert the keystrokes to integers.
    static int getNumber ( ) throws IOException
    {
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter an integer: ");
        String s = input.readLine();
        return Integer.parseInt(s);
    }

    //a main method to begin execution

    public static void main(String args[]) {
        int numOne = 0, numTwo = 1, numThree = 0;

        //the getNumber() method call must be enclosed in a try block because it is
        //defined with a throws clause
        try{
            numOne = getNumber();
            numTwo = getNumber();
            numThree = numOne / numTwo;
        }
//a catch statement that will receive the IO Exception and print the problem
catch (Exception e) {
    System.out.println("["] + e + "]");
}

System.out.println( numOne + "/" + numTwo + " = " + numThree);

Task Number 105

Identify the consequences of an unhandled exception.

Definition

Identification should include that improper exception handling will not

• allow a program that incurs an error to transfer control
• allow the program to continue running
• alert the user to the problem.

Task Number 106

Use multiple catch statements.

Definition

Using multiple catch statements should include

• defining at the end of the try block
• enabling the JVM to match the type of exception thrown to the parameters in the catch statement.

Task Number 107

Nest try blocks.

Definition
Nesting try blocks should be

- performed when an exception generated within the inner try block that is not caught by a catch is propagated to the outer try block
- continued up the stack trace until the exception is caught by the appropriate handler.

An example of a nested try block follows:

```java
//Use a nested try block

class NestTrys {
    public static void main (String args[ ]) {
        //Here, numer is longer than denom

        int numer[] = {4, 8, 16, 32, 64, 128, 256, 512};
        int denom[] = {2, 0, 4, 4, 0, 8};

        try {
            //outer try block
            for(int i = 0;  i < numer.length;  i++)  {
                try{
                    //the inner try block
                    System.out.println(numer[i] + "/" +
                    denom[i] + " is " +
                    numer[i] /denom[i]);
                }
                catch (ArithmeticException exc)  {
                    //some code to tell the user what happened to cause the error
                    System.out.println("Can’t divide by Zero!");
                }
            }
            }catch (ArrayIndexOutOfBoundsException exc) {
                the exception if the program tries to access an array index that doesn’t exist
                //the code is specifically tailored to match the purpose of the program
                System.out.println("No matching element found.");
                System.out.println("Fatal error - - program terminated.");
            }
        }
    }
}
```

---
Task Number 108

Throw an exception.

Definition

Throwing an exception involves manually throwing a statement that is not already automatically thrown by the JVM. The general format for the throw clause is:

throw exceptOb;

In this case, the exceptOb must be an object of an exception class derived from Throwable.

Task Number 109

Use the methods of Throwable.

Definition

Usage of the methods of Throwable requires that the class be extended and that a subclass of Throwable will have access to the methods of its superclass. Throwable has four constructors:

- Throwable()
- Throwable(String message)
- Throwable(String message, Throwable cause)
- Throwable(Throwable cause)

Also, as a superclass, Throwable has several methods which are available to all classes which inherit from it:

- getMessage()—Returns a message associated with the exception/error
- toString()—Returns a String containing a description of the Throwable object
- printStackTrace()—Prints the string return by toString() followed by the execution stack trace showing the sequence of a method invocations that led to the exception being thrown

Task Number 110

Create an original exception class.
Definition

Creating and defining an original exception class follows two general steps:

1. Define the class as a subclass of exception (name of class should end with "exception").
2. Provide constructors corresponding to exception’s 2 constructor, simply invoking super(..) in each case.

Sample code for creating an original exception class follows:

class NonIntResultException extends Exception {
    int n;
    int d;

    NonIntResultException(int i, int j) {
        n = i;
        d = j;
    }

    public String toString() {
        return "Result of " + n + " / " + d + 
               " is non-integer.";
    }
}

class CustomExceptDemo {
    public static void main(String args[]) {

        // Here, numer contains some odd values.
        int numer[] = { 4, 8, 15, 32, 64, 127, 256, 512 };  
        int denom[] = { 2, 0, 4, 4, 0, 8 }; 
        for(int i=0; i<numer.length; i++) {
            try {
                if((numer[i]%2) != 0)
                    throw new NonIntResultException(numer[i], denom[i]);

                System.out.println(numer[i] + " / " +
                                   denom[i] + " is " +
                                   numer[i]/denom[i]);
            }
            catch ( ArithmeticException exc) {
                // catch the exception
                System.out.println("Can't divide by Zero!");
            }
            catch ( ArrayIndexOutOfBoundsException exc) {

        }
// catch the exception
System.out.println("No matching element found.");
} catch (NonIntResultException exc) {
    System.out.println(exc);
}
}

Exploring Database Input/Output

Task Number 111

Describe a stream.

Definition

Description should identify a stream as

- an abstraction that either produces or consumes information
- that which is linked to a physical device by the Java I/O system.

I/O classes and methods may be applied to any type of device (the same methods used to write to the console can also be used to write to a disk file).

Task Number 112

List the main byte and character stream classes.

Definition

Listing main byte stream classes should include

- two class hierarchies: one for input and one for output
- two abstract classes at the top of each class hierarchy (from which are derived concrete subclasses that handle details of reading from and writing to various devices):
  - InputStream—defines characteristics common to byte input streams
  - OutputStream—defines characteristics common to byte output stream.
Listing character stream classes should include

- two class hierarchies: one for input and one for output
- two abstract classes, at the top of input and output hierarchies, whose concrete subclasses operate on Unicode character streams:
  - Reader is used for input.
  - Writer is used for output.

In general, the character-based classes parallel the byte-based classes.

**Task Number 113**

**Use predefined streams.**

**Definition**

Using predefined streams should include java.lang's three predefined stream variables: `System.in`, `System.out`, and `System.err` (defined as public and static and used by any part of program without reference to a specific System object). Terms are defined as follows:

- `System.out` (object of type PrintStream)
  - Refers to standard output stream
  - By default, this is the console.
- `System.in` (an object of type InputStream)
  - Refers to standard input
  - By default, this is the keyboard.
- `System.err` (object of type PrintStream)
  - Refers to standard error stream
  - By default, this is the console.

**Task Number 114**

**Read input from the keyboard.**

**Definition**

Reading input should be done from keystrokes at the command line of type String, as in the method declaration:

```java
public static void main(String args[]) {
}
```

**Task Number 115**
Write output to the monitor.

Definition

Written output is most easily performed with PrintStream methods:

- print()
- println()

PrintStream also implements OutputStream’s low-level method write( ), which can be used to write to the console.

Task Number 116

Use the byte streams for file I/O.

Definition

Using byte streams for file I/O should include identifying the abstract classes InputStream and OutputStream at the top of the inheritance hierarchies that read and write bytes. The general process follows:

- Open a stream.
- Write information.
- Read information.
- Close the stream.

Task Number 117

Read and write binary data.

Definition

Reading and writing binary data should include the read and write methods. InputStream class defines the following methods for reading bytes:

- int read() throws IOException
- int read(byte b[]) throws IOException
- int read(byte b[], int offset, int length) throws IOException

OutputStream class defines the following methods for writing bytes:

- void write(int b) throws IOException
• void write(byte b[]) throws IOException
• void write(byte b[], int offset, int length) throws IOException

Task Number 118

Use the character-based streams for console I/O.

Definition

Using character-based streams should include reading and writing characters to and from InputStream and OutputStream class hierarchies.

Task Number 119

Use the character-based streams for file I/O.

Definition

Using character-based streams for file I/O should include reading characters using the FileReader class. Sample code follows:

```java
//Create a FileReader class from the file name.
FileReader fr = new FileReader("filename.txt");
int i = fr.read(); //Read a character
```

Task Number 120

Create and manipulate streams in Java.

Definition

Creation of strings should include

• using new and the string constructor
• constructing one string object from another string object
• initializing a string variable to a character sequence.

Manipulation of strings should include

• concatenating (i.e., joining together) two strings by using the "+" operator
• arraying a string.
Understanding Multithreaded Programming Support

Task Number 121

Describe the fundamentals of multithreading.

Definition

Description of multithreading should include

- defining a *thread object* as one that encapsulates an executable instance
- listing several methods (e.g., final String getName( ); final int getPriority( ); final boolean isAlive( ); final void join( ); void run( ); void start( ))
- explaining the functions of high-priority threads and synchronization.

Task Number 122

Create multiple threads.

Definition

Creation of multiple threads should include

- adding more instances of a thread, each of which establishes an independent thread of execution
- identifying Java's two means for determining whether a thread has ended: isAlive( ) and join( )
- understanding that all created threads share the central processing unit (CPU) processing time and that Java is free to schedule each thread in its own way.

Task Number 123

Describe the life cycle of a thread.

Definition
Description of the life cycle should include the following thread states:

- Running—An active program is in progress.
- Ready to Run—The program will run as soon as it gets CPU time.
- Suspended—A program has been temporarily halted.
- Resumed—A suspended thread has been restarted.
- Blocked—A thread is waiting for a resource.
- Terminated—The thread execution has ended and cannot be resumed.

Task Number 124

Change a thread's priority.

Definition

Changing a thread’s priorities should include:

- calling setPriority( ) (i.e., final void setPriority(int level)
- designating level (must be within the range MIN_PRIORITY and MAX_PRIORITY)
- returning to default priority by specifying NORM_PRIORITY
- obtaining the Current Priority Setting by Calling getPriority( ).

Code to change the priority should include the following:

class Priority implements Runnable {
    int count;
    Thread thrd;

    static boolean stop = false;
    static String currentName;

    /* Construct a new thread. Notice that this
    constructor does not actually start the
    threads running. */
    Priority(String name) {
        thrd = new Thread(this, name);
        count = 0;
        currentName = name;
    }

    // Begin execution of new thread.
    public void run() {
        System.out.println(thrd.getName() + " starting.");
    }
}
```java
do {
    count++;
    
    if(currentName != thrd.getName()) {
        currentName = thrd.getName();
        System.out.println("In " + currentName);
    }
}

} while(stop == false && count < 1000);
stop = true;
System.out.println("\n" + thrd.getName() + " terminating.");

}
}
class PriorityDemo {
    public static void main(String args[]) {
        Priority mt1 = new Priority("High Priority");
        Priority mt2 = new Priority("Low Priority");

        // set the priorities
        mt1.thrd.setPriority(Thread.NORM_PRIORITY+2);
        mt2.thrd.setPriority(Thread.NORM_PRIORITY-2);

        // start the threads
        mt1.thrd.start();
        mt2.thrd.start();

        try {
            mt1.thrd.join();
            mt2.thrd.join();
        } catch(InterruptedException exc) {
            System.out.println("Main thread interrupted.");
        }

        System.out.println("\nHigh priority thread counted to " + mt1.count);
        System.out.println("Low priority thread counted to " + mt2.count);
    }
}
```

Task Number 125
Describe the purpose of synchronization.

Definition

Description of synchronization should include the following:

- Synchronization is a process of coordinating the activities of two or more threads.
- Java has a complete subsystem devoted to synchronization.
- Synchronization is most commonly needed when two or more threads need access to a shared resource that can be used by only one thread at a time.
- Synchronization is used when one thread is waiting for an event that is caused by another thread.
- All objects can be synchronized
- The key to synchronization is the concept of the monitor, which controls access to an object and implements the concept of a lock.

Task Number 126

Create synchronized methods.

Definition

Creation of a synchronized method should include the modifier synchronized (synchronized (obj) { statements }, which

- obtains the lock on obj before executing statements in block
- releases the lock when the statements block completes.

Task Number 127

Perform thread communication.

Definition

Performance of thread communication, supported by the methods wait(), notify(), and notifyAll(), should be performed when one thread notifies another of specific conditions (e.g., that it is blocked, that it can resume execution).

Task Number 128
Suspend, resume, and stop threads.

Definition

Threads should be designed so that the run( ) method periodically checks to see if the thread should suspend, resume, or stop its own execution, which is usually accomplished by establishing two-flag variables, one flag for suspend and resume, and one flag for stop, and the following rules:

- If set to running, run( ) must continue to let the thread execute.
- If set to suspend, the thread must pause.
- If set to stop, the thread must terminate.

Creating Applets, Events, and Other Topics

Task Number 129

Create a Java applet.

Definition

Creation of a basic applet must comply with the instructor's guidelines and involve written, executable, functioning code. An example of such an applet follows:

```java
// A minimal applet.
import java.awt.*;
import java.applet.*;

class SimpleApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Java makes applets easy.", 20, 20);
    }
}
```

// A minimal applet.
import java.awt.*;
import java.applet.*;

```html
/*
<applet code="SimpleApplet" width=200 height=60>
</applet>
*/
```
public class SimpleApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Java makes applets easy.", 20, 20);
    }
}

Task Number 130

Explain applet fundamentals, including its architecture.

Definition

Explanation of applet fundamentals should include the following:

- Applets use the init method to begin execution.
- Applets are small programs designed for transmission over the Internet and run within the browser.
- Applets can be executed by a program in the Java Development Kit called AppletViewer (as long as the applet code tag is embedded in the opening comments of a program).
- Applets use the event handling mode. (Events are the way an applet receives input from the outside world.)
- The four basic methods that may override, init(), start(), stop(), and destroy(), and a fifth method, paint(), are inherited from the AWT Component class and must be implemented.
- Applets need not override methods they do not use.

Task Number 131

Handle the passing of parameters into an applet from an HTML page.

Definition

Passing parameters to an applet should include using the PARAM attribute of the APPLET tag

- to specify the parameter’s name and value
- to retrieve a parameter, using the applet’s getParameter() 

and confirming that returned values and that any conversions are valid.

Actual code incorporating passing parameters follows:
// Pass a parameter to an applet.
import java.awt.*;
import java.applet.*;

/*
<applet code="Param" width=300 height=80>
<param name=author value="Herb Schildt">
<param name=purpose value="Demonstrate Parameters">
<param name=version value=2>
</applet>
*/

public class Param extends Applet {
    String author;
    String purpose;
    int ver;

    public void start() {
        String temp;

        author = getParameter("author");
        if(author == null) author = "not found";

        purpose = getParameter("purpose");
        if(purpose == null) purpose = "not found";

        temp = getParameter("version");
        try {
            if(temp != null)
                ver = Integer.parseInt(temp);
            else
                ver = 0;
        } catch(NumberFormatException exc) {
            ver = -1; // error code
        }

    }

    public void paint(Graphics g) {
        g.drawString("Purpose: " + purpose, 10, 20);
        g.drawString("By: " + author, 10, 40);
        g.drawString("Version: " + ver, 10, 60);
    }
}

Task Number 132
Describe which methods are called by the browser and when they are called.

Definition

Description should include

- init—to initialize the applet each time it is loaded (or reloaded)
- start—to start the applet's execution, such as when the applet's loaded or when the user revisits a page that contains the applet
- stop—to stop the applet's execution, such as when the user leaves the applet's page or quits the browser
- destroy—to perform a final cleanup in preparation for unloading.

Task Number 133

Describe procedures for connecting the Java applet to a database.

Definition

Description should include the three common methods for connecting a Java applet to a database:

- Thin JDBC
- HTTP with XML
- JavaBeans

NBEA Achievement Standards for Information Technology

Code a program solution in more than one programming language.

Task Number 134

Create new projects and applications in a development environment.

Definition
Creation of new projects should be performed in integrated development environment (IDE), such as Jdeveloper, to the instructor's requirements.

NBEA Achievement Standards for Information Technology

Choose the appropriate language or application development tool for specific tasks.

Test, debug, and document code.

Use application development tools to create code.

Task Number 135

Design and create a program to meet user needs.

Definition

Design and creation of a program may include

- performing a case study
- keeping a notebook or journal showing progress of the design
- interviewing end users to ensure compatibility
- fully commenting code
- testing operations
- giving demonstrations.

Creation of a business-related application should use Java elements, such as

- a main method
- at least two class files
- inheritance
- program flow control
- usage of the API
- overloaded methods
- overridden methods
- access modifiers to encapsulate code.

NBEA Achievement Standards for Information Technology

Choose the appropriate language or application development tool for specific tasks.

Identify and define the coding task.
Using Databases with Java

Task Number 136

Describe Java's relationship to Oracle's database.

Definition

Description should include the following:

- The Java programming language is complemented by tools that provide an end-to-end solution for creating, deploying, and managing Java applications.
- The total solution consists of client-side and server-side programmatic interfaces, tools to support Java development and the JVM, integrated with the Oracle database.
- All solutions are compatible with the Java standards.
- In addition to the JVM, the Java programming environment provides
  - Java-stored procedures, which are tightly integrated with PL/SQL, as the Java equivalent and companion for PL/SQL
  - the ability to call Java-stored procedures from PL/SQL packages and PL/SQL procedures from Java-stored procedures
  - the JDBC and SQLJ programming interfaces for accessing SQL data.

Task Number 137

List the Java-related driver classes for common databases.

Definition

The list should include

- Java-stored procedures
- SQLJ
- JDBC.

The following table is provided to aid the decision-making process for the appropriate use of Java API:
<table>
<thead>
<tr>
<th>Functionality Needed</th>
<th>Java API Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>To have a Java procedure called from SQL, such as a trigger.</td>
<td>Java-stored procedures</td>
</tr>
<tr>
<td>To call a static, simple SQL statement from a known table with known column names</td>
<td>SQLJ</td>
</tr>
<tr>
<td>from a Java object.</td>
<td></td>
</tr>
<tr>
<td>To call dynamic, complex SQL statements from a Java object.</td>
<td>JDBC</td>
</tr>
</tbody>
</table>

**Task Number 138**

**Describe the function of the JDBC package.**

**Definition**

Description should include the following:

- JDBC represents Java database connectivity.
- JDBC is the standard Java API for database-independent connectivity between the Java programming language and many databases.
- The JDBC library contains APIs for
  - making a connection to the database
  - creating SQL statements
  - executing SQL statements in the database
  - viewing and modifying the ResultSet (returned rows).

**Task Number 139**

**Describe the JDBC connection object.**

**Definition**

Description should include the following:

- The JDBC connection object is the gateway for communication to the database.
- All communication will pass through the JDBC connection object.

**Task Number 140**

**Analyze the process to connect to a database using JDBC package.**

**Definition**
Analysis should be performed on the programming process to establish a JDBC connection and includes

- importing the JDBC packages using `import` statements in the Java program to import the related Java classes
- registering the JDBC driver to instruct the JVM to load the driver into memory so JDBC requests can be performed
- creating database URL notation by creating a properly formatted URL address so the database can connect, such as in the following example:

```java
getConnection(String url)
getConnection(String url, String user, String password)
```

- creating a connection object by programming a call to the `DriverManager` object’s `getConnection()` method to establish the actual database connection. The most commonly used form of `getConnection()` requires you to pass a database URL, a username, and a password
- establishing a port-addressing connection, such as in the following summary and complete examples:

(Summary)

`jdbc:oracle:thin:@jdoe:1511:EMP URL Connection: DriverManager.getConnection(String url, Properties info);`

(Complete)

```java
import java.util.*;

String URL = "jdbc:oracle:thin:@jdoe:1511:EMP";
Properties info = new Properties();
info.put( "user", "username" );
info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);
```

FBLA Competitive Events and Activities Areas

Database Design & Applications

---

**Task Number 141**

**Describe the JDBC Statement object.**
Definition

Description should include the procedures for defining methods and properties that enable PL/SQL commands to send and receive SQL from a database through the following interfaces:

- JDBC Statement
- CallableStatement
- PreparedStatement

The following table provides a summary of each interface's purpose and recommended use:

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>Recommended Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
<td>Use for general-purpose access to a database. Useful when using static SQL statements at runtime. The Statement interface cannot accept parameters.</td>
</tr>
<tr>
<td>CallableStatement</td>
<td>Use when accessing database stored procedures. The CallableStatement interface can accept runtime input parameters.</td>
</tr>
<tr>
<td>PreparedStatement</td>
<td>Use when planning to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.</td>
</tr>
</tbody>
</table>

Task Number 142

Describe the use of Java code to iterate through the returned ResultSet.

Definition

Description should include the following:

- The SELECT statement is the standard way to select rows from a database and view them in a result set.
- A ResultSet object contains a table of data representing a database result set, which is generated by executing a statement that queries the database.
- The java.sql.ResultSet interface represents the result set of a database query.
- A ResultSet object maintains a cursor that points to the current row in the result set.
- The term result set refers to the row and column data contained in a ResultSet object.
- The methods of the ResultSet interface can be broken down into three categories:
  - Navigational methods are used to move the cursor around.
  - Get methods are used to view the data in the columns of the current row being pointed to by the cursor.
  - Update methods are used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well. The cursor is movable based on the properties of the ResultSet. These properties are designated when the corresponding statement that generated the ResultSet is created.
JDBC provides following connection methods to create statements with desired ResultSet:

createStatement(int RSType, int RSConcurrency);
prepareStatement(String SQL, int RSType, int RSConcurrency);
prepareCall(String sql, int RSType, int RSConcurrency);

The first argument above indicates the type of ResultSet object, and the second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable.

- The possible ResultSet types include the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResultSet.TYPE_FORWARD_ONLY</td>
<td>The cursor can only move forward in the result set.</td>
</tr>
<tr>
<td>ResultSet.TYPE_SCROLL_INSENSITIVE</td>
<td>The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.</td>
</tr>
<tr>
<td>ResultSet.TYPE_SCROLL_SENSITIVE</td>
<td>The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.</td>
</tr>
</tbody>
</table>

If the ResultSet type is not specified, TYPE_FORWARD_ONLY will be automatically provided.

FBLA Competitive Events and Activities Areas

Computer Problem Solving

Database Design & Applications

---

Task Number 143

Review a Java application that will pass variable and user input into a query.

Definition

Review should be made, as an example, to the following code that will pass a variable and user input into a query:

```java
String sql = "SELECT StudentID, FirstName, LastName FROM STUDENT";
ResultSet rs = stmt.executeQuery(sql);
//STEP 5: Extract data from result set
while(rs.next()){
```
//Retrieve by column name
int id = rs.getInt("StudentID ");
String first = rs.getString("FirstName ");
String last = rs.getString("LastName ");

//Display values
System.out.print("ID: " + StudentID);
System.out.print(" , First Name: " + FirstName);
System.out.println(" , Last Name: " + LastName);
}

FBLA Competitive Events and Activities Areas

Computer Problem Solving

Database Design & Applications

---

**Task Number 144**

**Describe the process to identify and handle Oracle exceptions in Java.**

**Definition**

Description should include the following:

- Exception handling allows for handling of predefined errors with predefined solutions.
- When an exception occurs, the exception is thrown. The program execution will stop, and
  the error/exception will pass to the nearest catch clause. If no catch clause is included, the
  program stops entirely.
- The most common exception in JDBC is java.sql.SQL; however,
  - a SQLException can occur in both the driver and the database
  - when an exception occurs, the SQLException object is passed to the catch clause
  - the SQLException object has several methods to get the error code, get the
    message, and print the error.

FBLA Competitive Events and Activities Areas

Database Design & Applications
**Task Number 145**

**Review a Java application to write data to a database table.**

**Definition**

Review should be made, as an example, to the following code for writing data from within Java to a database table:

```java
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExampleWrite {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.oracle.jdbc.Driver";
    static final String DB_URL = "jdbc:oracle://localhost/CourseSelection";

    // Username and Password
    static final String USER = "username";
    static final String PASS = "password";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        try{
            //STEP 2: Register JDBC driver
            Class.forName("com.oracle.jdbc.Driver");

            //STEP 3: Open a connection
            System.out.println("Connecting to an oracle database...");
            conn = DriverManager.getConnection(OracleURL, USER, PASS);
            System.out.println("Connected to database successfully...");

            //STEP 4: Execute a query
            System.out.println("Inserting records into the CourseSelection table...");
            stmt = conn.createStatement();

            String sql = "INSERT INTO CourseSelection " + "VALUES (6660, 'Drake', 'Thane', 18)";
            stmt.executeUpdate(sql);
            sql = "INSERT INTO CourseSelection "+ "VALUES (6630, 'Peyton', 'Willem', 25)";
            stmt.executeUpdate(sql);
            sql = "INSERT INTO CourseSelection " + "VALUES (6640, 'Kim', 'Grace', 30)";
            stmt.executeUpdate(sql);
            sql = "INSERT INTO CourseSelection " + "VALUES(6662, 'Evers', 'Steph', 28)";
            stmt.executeUpdate(sql);
            System.out.println("Inserted data into the table...");
        }
    }
}
```
}catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
}catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
}finally{
    //finally block used to close resources
    try{
        if(stmt!=null)
            conn.close();
    }catch(SQLException se){
        // do nothing
    }catch(SQLException se){
        se.printStackTrace();
    }
} //end finally try
} //end try
System.out.println("Success!");
} //end main
} //end JDBCExampleWrite

: 

FBLA Competitive Events and Activities Areas

Computer Problem Solving

Database Design & Applications

Preventing for Industry Certification

Task Number 146
Describe the process and requirements for obtaining industry certifications related to the Java Programming (Oracle) course.

Definition

The description should include a list of industry certifications related to the Java Programming (Oracle), course and the process/requirements for obtaining the certifications from

- official websites of the testing organization/vendor
- practice materials and tests based on information from the testing organization/vendor
- information from certified instructors or industry-certified professionals
- information in the "Course Description" section of this document.

FBLA Competitive Events and Activities Areas

Computer Applications

Job Interview

NBEA Achievement Standards for Information Technology

Obtain programming industry certification(s) needed for chosen career path.

Task Number 147

Identify testing skills and strategies for a certification examination.

Definition

The identification of testing skills and strategies should be undertaken by

- conducting an Internet research project
- reviewing materials from exam and practice-exam publishers
- interviewing certified instructors and/or industry-certified professionals.

FBLA Competitive Events and Activities Areas

Computer Applications
NBEA Achievement Standards for Information Technology

Obtain programming industry certification(s) needed for chosen career path.

Task Number 148

Demonstrate ability to successfully complete selected practice examinations (e.g., practice questions similar to those on certification exams).

Definition

Demonstration should include successfully completing practice examinations for selected certifications related to the course obtained from vendor sites and/or materials from publishers. The level of performance on a practice examination serves as a gauge of the applicant's readiness for formal industry testing.

FBLA Competitive Events and Activities Areas

Computer Applications

Job Interview

NBEA Achievement Standards for Information Technology

Obtain programming industry certification(s) needed for chosen career path.

Task Number 149

Complete an industry certification examination representative of skills learned in this course (e.g., MOS, MTA, IC3).
**Definition**

Completion will be achieved when the student applicant earns an examination score deemed "passing" by the testing organization. Qualifying examinations are those currently approved at the state level as representative of Java Programming (Oracle) skills.

Students should be encouraged to attain industry certification as evidence of their database design and management skill level and general employability.

---

**SOL Correlation by Task**

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>English:</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>Compare PL/SQL and Java.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>40</td>
<td>Identify the advantages of using PL/SQL in database application development.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>41</td>
<td>Identify the advantages of using Java in database application development.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>42</td>
<td>Describe Java's place in computer-language history.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>43</td>
<td>Explain the fundamentals of object-oriented programming.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>44</td>
<td>Describe the general form of a Java program, including variables, program flow control, main, and methods statements.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>45</td>
<td>Create and compile a block of code.</td>
<td>COM.5</td>
</tr>
<tr>
<td>46</td>
<td>Identify Java keywords.</td>
<td>11.3, 11.5, 12.3, 12.5</td>
</tr>
<tr>
<td>47</td>
<td>Describe the standard Java naming conventions.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>48</td>
<td>Identify the key components of the Software Development Kit (SDK).</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>49</td>
<td>Describe the JVM.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>50</td>
<td>Use Java's primitive data types.</td>
<td>COM.14, COM.15</td>
</tr>
<tr>
<td>51</td>
<td>Initialize and assign variables.</td>
<td>COM.15</td>
</tr>
<tr>
<td>52</td>
<td>Describe the scope rules of a variable.</td>
<td>11.5, 12.5</td>
</tr>
<tr>
<td>53</td>
<td>Apply type conversion (casting) in expressions.</td>
<td>COM.16</td>
</tr>
<tr>
<td>Page</td>
<td>Task Description</td>
<td>Mathematics</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>54</td>
<td>Use the arithmetic operators.</td>
<td>COM.1, COM.7, COM.15</td>
</tr>
<tr>
<td>55</td>
<td>Compare relational and logical operators.</td>
<td>English: 11.5, 12.5 Mathematics: COM.1, COM.7, COM.8, COM.15, COM.16</td>
</tr>
<tr>
<td>56</td>
<td>Input characters from the keyboard.</td>
<td>Mathematics: COM.13</td>
</tr>
<tr>
<td>57</td>
<td>Navigate the Java API.</td>
<td>Mathematics: COM.2</td>
</tr>
<tr>
<td>58</td>
<td>Build and execute Java applications, using a development tool.</td>
<td>Mathematics: COM.2</td>
</tr>
<tr>
<td>59</td>
<td>Modify Java application source code.</td>
<td>Mathematics: COM.5</td>
</tr>
<tr>
<td>60</td>
<td>Identify situations in which to use control statements.</td>
<td>English: 11.5, 12.5 Mathematics: COM.8</td>
</tr>
<tr>
<td>61</td>
<td>Use if-then and if-then-else statements.</td>
<td>English: 11.5, 12.5 Mathematics: COM.8</td>
</tr>
<tr>
<td>62</td>
<td>Use switch statements.</td>
<td>Mathematics: COM.8</td>
</tr>
<tr>
<td>63</td>
<td>Use for loop statements.</td>
<td>Mathematics: COM.13</td>
</tr>
<tr>
<td>64</td>
<td>Use the while loop.</td>
<td>Mathematics: COM.8, COM.13</td>
</tr>
<tr>
<td>65</td>
<td>Use the do-while loop.</td>
<td>Mathematics: COM.13</td>
</tr>
<tr>
<td>66</td>
<td>Use break and continue statements effectively.</td>
<td>Mathematics: COM.13</td>
</tr>
<tr>
<td>67</td>
<td>Identify class and method definitions.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>68</td>
<td>Differentiate between classes and objects.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>69</td>
<td>Describe an object and its operations (or methods).</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>70</td>
<td>Create a method, and return a value from a method.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>71</td>
<td>Use parameters in a method.</td>
<td>Mathematics: COM.5, COM.16</td>
</tr>
<tr>
<td>72</td>
<td>Add a constructor to a class.</td>
<td>Mathematics: COM.5</td>
</tr>
<tr>
<td>73</td>
<td>Overload a constructor.</td>
<td>Mathematics: COM.14</td>
</tr>
<tr>
<td>74</td>
<td>Describe garbage collection and the finalize () method.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>75</td>
<td>Apply the this reference.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>76</td>
<td>Explain the purpose and importance of the Object class.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>77</td>
<td>Use single and multidimensional arrays.</td>
<td>Mathematics: COM.5</td>
</tr>
<tr>
<td>78</td>
<td>Use various formats to initialize arrays.</td>
<td>Mathematics: COM.14</td>
</tr>
<tr>
<td>79</td>
<td>Apply the bitwise operators.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>80</td>
<td>Create objects of type String and utilize their methods.</td>
<td>Mathematics: COM.5</td>
</tr>
</tbody>
</table>
| 81 | Define the public- and private-access modifiers. | English: 11.3, 12.3  
Mathematics: COM.11 |
| 82 | Pass and return objects/primitive parameters to methods. | Mathematics: COM.5 |
| 83 | Use class variables and class methods. | Mathematics: COM.15 |
| 84 | Apply recursive methods. | Mathematics: COM.9 |
| 85 | Create and use recursive methods and variables. | English: 11.3, 12.3  
Mathematics: COM.9 |
| 86 | Describe nested and inner classes. | English: 11.5, 12.5  
Mathematics: COM.14 |
| 87 | Use inheritance to define new classes. | Mathematics: COM.14,  
COM.16 |
| 88 | Describe the difference between a superclass and a subclass. | English: 11.5, 12.5  
Mathematics: COM.5,  
COM.16 |
| 89 | Explain how inheritance affects member access. | English: 11.5, 12.5 |
| 90 | Use super to call superclass constructors and members. |  |
| 91 | Create a multilevel class hierarchy. | Mathematics: COM.14 |
| 92 | Explain cases when constructors are called in a class hierarchy. | English: 11.5, 12.5 |
| 93 | Apply superclass references to subclass objects. | English: 11.5, 12.5  
Mathematics: COM.5 |
| 94 | Demonstrate how to override methods. | English: 11.5, 11.6, 12.5,  
12.6 |
| 95 | Create abstract methods and classes. | English: 11.1, 12.1  
Mathematics: COM.15 |
| 96 | Use the keyword final to prevent inheritance. | English: 11.5, 12.5 |
| 97 | Create and implement an interface, utilizing its variables. | Mathematics: COM.10 |
| 98 | Extend an interface. | Mathematics: COM.10 |
| 99 | Create and import a package. | Mathematics: COM.7 |
| 100 | Describe how a package affects access to its programs. | English: 11.5, 12.5  
Mathematics: COM.16 |
| 101 | Explain the purpose of the Thread class and the Runnable interface. | English: 11.5, 12.5 |
| 102 | Explain the purpose of exception handling. | English: 11.3, 11.5, 12.3,  
12.5 |
| 103 | Write code to demonstrate how try and catch work together to handle an exception. | English: 11.5, 11.6, 11.7,  
12.5, 12.6, 12.7 |
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>Create an exception handler.</td>
<td>Mathematics: COM.16</td>
</tr>
<tr>
<td>105</td>
<td>Identify the consequences of an unhandled exception.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>106</td>
<td>Use multiple <em>catch</em> statements.</td>
<td>Mathematics: COM.16</td>
</tr>
<tr>
<td>107</td>
<td>Nest <em>try</em> blocks.</td>
<td>English: 11.3, 12.3</td>
</tr>
<tr>
<td>108</td>
<td>Throw an exception.</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>Use the methods of Throwable.</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Create an original exception class.</td>
<td>English: 11.3, 12.3</td>
</tr>
<tr>
<td>111</td>
<td>Describe a stream.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>112</td>
<td>List the main byte and character stream classes.</td>
<td>English: 11.5, 11.6, 11.7, 12.5, 12.6, 12.7</td>
</tr>
<tr>
<td>113</td>
<td>Use predefined streams.</td>
<td>Mathematics: COM.15</td>
</tr>
<tr>
<td>114</td>
<td>Read input from the keyboard.</td>
<td>Mathematics: COM.7</td>
</tr>
<tr>
<td>115</td>
<td>Write output to the monitor.</td>
<td>Mathematics: COM.10</td>
</tr>
<tr>
<td>116</td>
<td>Use the byte streams for file I/O.</td>
<td>English: 11.6, 11.7, 12.6, 12.7</td>
</tr>
<tr>
<td>117</td>
<td>Read and write binary data.</td>
<td>Mathematics: COM.11</td>
</tr>
<tr>
<td>118</td>
<td>Use the character-based streams for console I/O.</td>
<td>Mathematics: COM.15</td>
</tr>
<tr>
<td>119</td>
<td>Use the character-based streams for file I/O.</td>
<td>Mathematics: COM.15</td>
</tr>
<tr>
<td>120</td>
<td>Create and manipulate streams in Java.</td>
<td>Mathematics: COM.2</td>
</tr>
<tr>
<td>121</td>
<td>Describe the fundamentals of multithreading.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>122</td>
<td>Create multiple threads.</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>Describe the life cycle of a thread.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>124</td>
<td>Change a thread's priority.</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>Describe the purpose of synchronization.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>126</td>
<td>Create synchronized methods.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>127</td>
<td>Perform thread communication.</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>Suspend, resume, and stop threads.</td>
<td>Mathematics: COM.2</td>
</tr>
<tr>
<td>129</td>
<td>Create a Java applet.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>130</td>
<td>Explain applet fundamentals, including its architecture.</td>
<td>Mathematics: COM.3</td>
</tr>
<tr>
<td>131</td>
<td>Handle the passing of parameters into an applet from an HTML page.</td>
<td>Mathematics: COM.3, COM.16</td>
</tr>
<tr>
<td></td>
<td>Describe which methods are called by the browser and when they are called.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>Mathematics: COM.16</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>Describe procedures for connecting the Java applet to a database.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td></td>
<td>Mathematics: COM.2</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>Create new projects and applications in a development environment.</td>
<td>Mathematics: COM.1</td>
</tr>
<tr>
<td>135</td>
<td>Design and create a program to meet user needs.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td></td>
<td>Mathematics: COM.1</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>Describe Java's relationship to Oracle's database.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>137</td>
<td>List the Java-related driver classes for common databases.</td>
<td>English: 11.6, 12.6</td>
</tr>
<tr>
<td></td>
<td>Mathematics: COM.15</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>Describe the function of the JDBC package.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>139</td>
<td>Describe the JDBC connection object.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>140</td>
<td>Analyze the process to connect to a database using JDBC package.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td></td>
<td>Mathematics: COM.5</td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>Describe the JDBC Statement object.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>142</td>
<td>Describe the use of Java code to iterate through the returned ResultSet.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td></td>
<td>Mathematics: COM.4</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>Review a Java application that will pass variable and user input into a query.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td></td>
<td>Mathematics: COM.17</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>Describe the process to identify and handle Oracle exceptions in Java.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td></td>
<td>Mathematics: COM.4</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>Review a Java application to write data to a database table.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>146</td>
<td>Describe the process and requirements for obtaining industry certifications related to the Java Programming (Oracle) course.</td>
<td>English: 11.5, 11.8, 12.5, 12.8</td>
</tr>
<tr>
<td>147</td>
<td>Identify testing skills and strategies for a certification examination.</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>148</td>
<td>Demonstrate ability to successfully complete selected practice examinations (e.g., practice questions similar to those on certification exams).</td>
<td>English: 11.5, 12.5</td>
</tr>
<tr>
<td>149</td>
<td>Complete an industry certification examination representative of skills learned in this course (e.g., MOS, MTA, IC3).</td>
<td></td>
</tr>
</tbody>
</table>

**Instructional Resources for Java**
Oracle® Database Java Developer's Guide 11g Release 1 (11.1) (See sidebar navigation for more.)

- http://docs.oracle.com/cd/B28359_01/java.111/b31225/index.htm#A

Java Ain't Scary, Introducing Java to PL/SQL Programmers (slideshow)

- http://www.slideshare.net/lucasjellema/java-aint-scary-introducing-java-to-plsql-developers

Java vs. PL/SQL: Where Do I Put the SQL?


Cyber Security and Cyber Forensics Infusion Units

Cyber Security and Cyber Forensics Infusion Units (CYBR) were designed to be infused with designated CTE courses to help students in those programs achieve additional, focused, validated tasks/competencies in personal and professional cyber security skills. These units are not mandatory, and, as such, the tasks/competencies are marked as "optional," to be taught at the instructor's discretion.

Entrepreneurship Infusion Units

Entrepreneurship Infusion Units may be used to help students achieve additional, focused competencies and enhance the validated tasks/competencies related to identifying and starting a new business venture. Because the unit is a complement to certain designated courses and is not mandatory, all tasks/competencies are marked “optional.”
Appendix: Credentials, Course Sequences, and Career Cluster Information

Industry Credentials: Only apply to 36-week courses

- Certified Internet Web (CIW) Advanced HTML 5 and CSS 3 Specialist Examination
- Certified Internet Web (CIW) Data Analyst Examination
- Certified Internet Web (CIW) Database Design Specialist Examination
- Certified Internet Web (CIW) E-Commerce Services Specialist Examination
- Certified Internet Web (CIW) Internet Business Associate Examination
- Certified Internet Web (CIW) JavaScript Specialist Examination
- Certified Internet Web (CIW) Network Technology Associate Examination
- Certified Internet Web (CIW) Site Development Associate Examination
- Certified Internet Web (CIW) Social Media Strategist Examination
- Certified Internet Web (CIW) User Interface Designer Examination
- Certified Internet Web (CIW) Web Design Specialist Examination
- Certified Internet Web (CIW) Web Security Specialist Examination
- Cloud Essentials Certification Examination
- College and Work Readiness Assessment (CWRA+)
- Computer Programming Assessment
- IC3 Digital Literacy Certification Examination
- Microsoft 365 Fundamentals Examination
- Microsoft Certified Azure Fundamentals Examination
- Microsoft Dynamics 365 Fundamentals Examination
- Microsoft Office Specialist (MOS) Examinations
- Microsoft Office Specialist (MTA) Examinations
- National Career Readiness Certificate Assessment
- Oracle Certified Associate Examinations
- Oracle Database Design and Programming with SQL Examination
- Oracle Programming with PL/SQL Examination
- Workplace Readiness Skills for the Commonwealth Examination

Concentration sequences: A combination of this course and those below, equivalent to two 36-week courses, is a concentration sequence. Students wishing to complete a specialization may take additional courses based on their career pathways. A program completer is a student who has met the requirements for a CTE concentration sequence and all other requirements for high school graduation or an approved alternative education program.

- Computer Information Systems (6612/36 weeks)
- Computer Information Systems (6614/18 weeks)
- Computer Information Systems, Advanced (6613/36 weeks)
- Computer Information Systems, Advanced (6615/18 weeks)
- Computer Network Software Operations (6650/36 weeks)
- Computer Network Software Operations, Advanced (6651/36 weeks)
- Cybersecurity Software Operations (6304/36 weeks)
- Database Design and Management (Oracle) (6660/36 weeks)
- Database Design and Management with PL/SQL (Oracle) (6662/36 weeks)
- Digital Applications (6611/36 weeks)
- Digital Applications (6617/18 weeks)
- Information Technology Fundamentals (6670/36 weeks)
- International Baccalaureate Information Technology in a Global Society (IB6613/36 weeks)
- Office Administration (6621/36 weeks)
- Office Administration (6622/18 weeks)
- Office Specialist I--Preparation (6740/36 weeks)
- Office Specialist II--Preparation (6741/36 weeks)
- Office Specialist III--Preparation (6742/36 weeks)
- Programming (6640/36 weeks)
- Programming, Advanced (6641/36 weeks)

### Career Cluster: Information Technology

<table>
<thead>
<tr>
<th>Pathway</th>
<th>Occupations</th>
</tr>
</thead>
</table>
| Information Support and Services | Computer Support Specialist  
Computer Systems Engineer, Architect  
Database Administrator  
Database Analyst  
Network Systems and Data Communication Analyst  
Software Test Engineer |
| Network Systems               | Computer and Information Systems Administrator  
Computer Support Specialist  
Computer Systems Engineer, Architect  
Database Analyst  
Network Systems and Data Communication Analyst  
Software Test Engineer |

### Career Cluster: Science, Technology, Engineering and Mathematics

<table>
<thead>
<tr>
<th>Pathway</th>
<th>Occupations</th>
</tr>
</thead>
</table>
| Engineering and Technology   | Computer Hardware Engineer  
Computer Programmer  
Computer Software Engineer  
Network and Computer Systems Administrator  
Network Systems and Data Communication Analyst  
Production, Planning, Expediting Clerk  
Project Manager  
Stockroom, Warehouse, or Storage Yard Stock Clerk  
Technical Writer  
Telecommunications Specialist  
Transportation Manager |