| [Print](javascript:window.print()) | [Close](javascript:parent.window.close()) |

**Course Transcript**

Java SE7 Fundamentals: Decision Constructs, Arrays, and Loops

**Java Operators, Decision Constructs, and Arrays**

| [1. Using Operators and Decision Constructs](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#t2) |

| [2. Creating and Using Arrays](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#t25) |

| [3. Working with Java Arrays](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#t26) |

**Using Loops in Java**

| [1. Introducing Loop Constructs](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#t14) |

| [2. Using Loops](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#t16) |

| [3. Using Loop Constructs and Arrays](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#t27) |

Using Operators and Decision Constructs

Learning Objectives

*After completing this topic, you should be able to*

* *use if and if/else constructs*
* *use a switch statement*

**1. Using if/else constructs**

In everyday life, you have to make a lot of decisions, and you often use the word “if” with some condition when making these decisions. For example, you may think "If the house is blue, I will take a tour of it." Or, "If the car is a sports car and it is safe, I will buy it." These types of decisions go through our minds subconsciously every day. One of the tasks that programs often perform is to evaluate a condition and, depending on the result, execute different blocks or branches of code.   
  
For example, your program might check to see if the value of one variable is equal to the value of another variable and, if so, do something.

Graphic

*For example, the program might execute a statement if the condition is true and continue evaluating the rest of the code if it's false.*

In addition to arithmetic operators, such as plus (+) and increment (++), the Java programming language provides several relational operators:

* less than
* greater than, and
* AND   
  *AND is represented by two ampersands.*

These operators are used when you want your program to execute different blocks or branches of code depending on different conditions, such as checking if the value of two variables is the same.

Note

*Each of these operators is used within the context of a decision construct, such as an if or if/else construct.*

For example, consider a program that controls an elevator with specific functionality:

Code

    public class Elevator {  
        public boolean doorOpen=false; // Default setting  
        public int currentFloor = 1; // Default starting point  
        public final int TOP\_FLOOR = 10;  
        public final int MIN\_FLOORS = 1;

* the doors of the elevator can open   
    
  **Code**  
      public class Elevator {  
          public boolean doorOpen=false; // Default setting  
          public int currentFloor = 1; // Default starting point  
          public final int TOP\_FLOOR = 10;  
          public final int MIN\_FLOORS = 1;  
    
          public void openDoor() {  
              System.out.println(“Opening door.”);  
              doorOpen = true;  
              System.out.println(“Door is open.”);  
          }  
     }
* the doors of the elevator can close   
    
  **Code**  
      public class Elevator {  
          public boolean doorOpen=false; // Default setting  
          public int currentFloor = 1; // Default starting point  
          public final int TOP\_FLOOR = 10;  
          public final int MIN\_FLOORS = 1;  
    
          public void closeDoor() {  
              System.out.println(“Closing door.”);  
              doorOpen = false;  
              System.out.println(“Door is closed.”);  
          }  
     }
* the elevator can go up one floor, and   
    
  **Code**  
      public class Elevator {  
          public boolean doorOpen=false; // Default setting  
          public int currentFloor = 1; // Default starting point  
          public final int TOP\_FLOOR = 10;  
          public final int MIN\_FLOORS = 1;  
    
          public void goUp() {  
              System.out.println(“Going up one floor.”);  
              currentFloor++;  
              System.out.println(“Floor: “ + currentFloor);  
         }  
     }
* the elevator can go down one floor   
    
  **Code**  
      public class Elevator {  
          public boolean doorOpen=false; // Default setting  
          public int currentFloor = 1; // Default starting point  
          public final int TOP\_FLOOR = 10;  
          public final int MIN\_FLOORS = 1;  
    
          public void goDown() {  
              System.out.println(“Going down one floor.”);  
              currentFloor--;  
              System.out.println(“Floor: “ + currentFloor);  
          }  
     }

A test class similar to this example, runs the elevator through some tests.

Code

    public class ElevatorTest {  
        public static void main(String args[]) {  
  
            Elevator myElevator = new Elevator();  
            myElevator.closeDoor();  
            myElevator.goDown();  
            myElevator.goUp();  
            myElevator.goUp();  
            myElevator.goUp();  
            myElevator.openDoor();  
            myElevator.closeDoor();  
            myElevator.goDown();  
            myElevator.openDoor();  
            myElevator.goDown();  
            myElevator.openDoor();  
        }  
   }

Relational operators compare two values to determine their relationship. The table lists the different conditions you can test by using relational operators. The result of all relational operators is a Boolean value. Boolean values can be either true or false. For instance, all the examples in the table yield a Boolean result of true.

Graphic

*The operator == is used to test for a value that is equal to another, as in this example:   
int i=1;  
(i == 1)  
The operator != is used to test for a value that is not equal to another, as in this example:  
int i=2;  
(i != 1)  
The operator < is used to test for a value that is less than another value, as in this example:  
int i=0;  
(i < 1)  
The operator <= is used to test for a value that is less than, or equal to, another value, as in this example:  
int i=1;  
(i <= 1)  
The operator > is used to test for a value that is greater than another value, as in this example:  
int i=2;  
(i > 1)  
The operator >= is used to test for a value that is greater than, or equal to, another value, as in this example:  
int i=1;  
(i >= 1)*

Note

*The equal sign is used to make an assignment.*

If you use the == operator to compare object references to String objects, the operator tests whether the address in the String object references in memory are equal, not their contents.  
  
If you want to test equality between the strings of characters, such as whether the name Fred Smith is equal to Joseph Smith, use the equals method of the String class. The class in this example contains two employee names and a method that compares names.

Code

    public class Employees {  
        public String name1 = "Fred Smith";  
        public String name2 = "Joseph Smith";  
  
        public void areNamesEqual() {  
  
            if (name1.equals(name2)) {  
                System.out.println("Same name.");  
            }  
            else {  
                System.out.println("Different name.");  
            }  
        }  
   }

You will also need to be able to make a single decision based on more than one condition. Under such circumstances, you can use conditional operators to evaluate complex conditions as a whole. This table lists the common conditional operators in the Java programming language.

Graphic

*The operator && is used to test for one condition and another condition, as in this example:   
int i = 2;  
int j = 8;  
((i < 1) && (j > 6))  
The operator || is used to test for either one condition or another condition, as in this example:  
int i = 2;  
int j = 8;  
((i < 1) || (j > 10))  
The operator ! is used to test for a condition that doesn't exist, as in this example:  
int i = 2;  
(!(i < 3))*

Note

*All the examples in the table yield a Boolean result of false.*

Question

Match each relational or conditional operator to the statement that expresses it.

**Options:**

1. ||
2. &&
3. !

**Targets:**

1. If the toy is red, I will purchase it. However, if the toy is yellow, I will also purchase it.
2. If the toy is yellow and costs the same as or more than another red item, I will  
   purchase it.
3. If the toy is green, I will not purchase it.

Answer

*The double pipe symbol operator expresses an action to be taken if either one condition or another condition exists.*

*The double ampersand operator expresses an action to be taken if one condition and another condition exist.*

*The exclamation mark operator expresses an action not to be taken if a condition exists.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

An if statement, or an if construct, executes a block of code if an expression is true. There are a few variations on the basic if construct.

Syntax

if (*boolean\_expression*) {  
    *code\_block*  
} // end of if construct  
// program continues here

*The boolean\_expression is a combination of relational operators, conditional operators, and values resulting in a value of true or false.*

*The code\_block represents the lines of code that are executed if the expression is true.*

First the boolean\_expression is tested. If the expression is true, the code block is executed.   
  
If the boolean\_expression is not true, the program skips to the end of the if construct code block.

Syntax

if (*boolean\_expression*) {  
    *code\_block*  
} // end of if construct  
// program continues here

The ElevatorTest class tests an Elevator object by invoking its methods. One of the first methods that the ElevatorTest class invokes is the goDown method. Two if statements can fix this problem.

Code

    ...  
    public void goDown() {  
              
            if (currentFloor == MIN\_FLOORS) {  
                System.out.println("Cannot Go down");  
            }  
            if (currentFloor > MIN\_FLOORS) {  
                System.out.println("Going down one floor.");  
                currentFloor--;  
                System.out.println("Floor: " + currentFloor);  
            }  
        }  
   }

This goDown method contains two if constructs that determine whether the elevator should go down or display an error:

Code

    ...  
    public void goDown() {  
              
            if (currentFloor == MIN\_FLOORS) {  
                System.out.println("Cannot Go down");  
            }  
            if (currentFloor > MIN\_FLOORS) {  
                System.out.println("Going down one floor.");  
                currentFloor--;  
                System.out.println("Floor: " + currentFloor);  
            }  
        }  
   }

* the program displays an error if the elevator cannot go down, and   
  *The relevant code is:  
    
  if (currentFloor == MIN\_FLOORS) {  
                 System.out.println("Cannot Go down");  
             }*   
    
  **Code**  
      ...  
      public void goDown() {  
                
              if (currentFloor == MIN\_FLOORS) {  
                  System.out.println("Cannot Go down");  
              }  
              if (currentFloor > MIN\_FLOORS) {  
                  System.out.println("Going down one floor.");  
                  currentFloor--;  
                  System.out.println("Floor: " + currentFloor);  
              }  
          }  
     }
* If the elevator can go down, the current floor plus the new floor are displayed   
  *The relevant code is:  
    
  if (currentFloor > MIN\_FLOORS) {  
                 System.out.println("Going down one floor.");  
                 currentFloor--;  
                 System.out.println("Floor: " + currentFloor);  
             }*   
    
  **Code**  
      ...  
      public void goDown() {  
                
              if (currentFloor == MIN\_FLOORS) {  
                  System.out.println("Cannot Go down");  
              }  
              if (currentFloor > MIN\_FLOORS) {  
                  System.out.println("Going down one floor.");  
                  currentFloor--;  
                  System.out.println("Floor: " + currentFloor);  
              }  
          }  
     }

Supplement

*Selecting the link title opens the resource in a new browser window.*

**Job aid**

Access the job aid [IfElevator class](javascript:doWindow('./ja_jsef_a03_it_enust203_frame.html')) to view the entire source code for this class – the program that the ElevatorTest class tests.

In this example output, the elevator logic prevents a problem.

Graphic

*The relevant line of output is Cannot Go down.*

Code

Opening door.  
Door is open.  
Closing door.  
Door is closed.  
Cannot Go down  
Going up one floor.  
Floor: 2  
Going up one floor.  
Floor: 3  
...

Sometimes you might need to execute an if statement as part of another if statement. The code example illustrates how to use nested if statements to check the values of two variables.   
  
If the value of the currentFloor variable is equal to the MIN\_FLOORS constant, an error message is printed and the elevator does not go down. If the value of the currentFloor variable is greater than the MIN\_FLOORS constant and the doors are closed, the elevator goes down.

Graphic

*The relevant code is:  
  
if (!doorOpen) {  
                   System.out.println("Going down one floor.");  
                   currentFloor--;  
                   System.out.println("Floor: " + currentFloor);  
               }*

Code

    ...  
    public void goDown() {  
  
            if (currentFloor == MIN\_FLOORS) {  
                System.out.println("Cannot Go down");  
            }  
  
            if (currentFloor > MIN\_FLOORS) {  
  
                if (!doorOpen) {  
                    System.out.println("Going down one floor.");  
                    currentFloor--;  
                    System.out.println("Floor: " + currentFloor);  
                }  
            }  
        }  
  
   }

Note

*Use nested if/else constructs sparingly because they can be confusing to debug.*

Often, you want one block of code to be executed if the expression is true, and another block of code to be executed if the expression is false. You can use an if construct to execute a code block if the expression is true with an else construct that executes only if the expression is false.

Syntax

if (*boolean\_expression*) {  
    *code\_block1*  
} // end of if construct  
else {  
    *code\_block2*  
} // end of else construct  
// program continues here

This is the syntax for the if/else construct.

Syntax

if (*boolean\_expression*) {  
    *code\_block1*} // end of if construct  
else {  
    *code\_block2*} // end of else construct  
// program continues here

*The boolean\_expression is a combination of relational operators, conditional operators, and values resulting in a value of true or false.*

*The code\_block1 represents the lines of code that are executed if the expression is true.*

*The code\_block2 represents the lines of code that are executed if the expression is false.*

You can use one if/else statement to fix the problem of the elevator going to an invalid floor. The goDown method in the example contains one if/else construct that determines whether the elevator should go down or display an error:

Code

    public void goUp() {  
            System.out.println("Going up one floor.");  
            currentFloor++;  
            System.out.println("Floor: " + currentFloor);  
        }  
  
        public void goDown() {  
  
            if (currentFloor == MIN\_FLOORS) {  
                System.out.println("Cannot Go down");  
            }  
            else {  
                System.out.println("Going down one floor.");  
                currentFloor--;  
                System.out.println("Floor: " + currentFloor);}  
            }  
        }  
   }

* If the value of the currentFloor variable is equal to the MIN\_FLOORS constant, an error message is printed and the elevator does not go down.   
  *The relevant code is:  
    
  public void goDown() {  
    
  if (currentFloor == MIN\_FLOORS) {  
  System.out.println("Cannot Go down");  
  }*   
    
  **Code**  
      public void goUp() {  
              System.out.println("Going up one floor.");  
              currentFloor++;  
              System.out.println("Floor: " + currentFloor);  
          }  
    
          public void goDown() {  
    
              if (currentFloor == MIN\_FLOORS) {  
                  System.out.println("Cannot Go down");  
              }  
              else {  
                  System.out.println("Going down one floor.");  
                  currentFloor--;  
                  System.out.println("Floor: " + currentFloor);}  
              }  
          }  
     }
* Otherwise – else – the value of the currentFloor variable is assumed to be greater than the MIN\_FLOORS constant and the elevator goes down.   
  *The relevant code is:  
    
  else {  
  System.out.println("Going down one floor.");  
  currentFloor--;  
  System.out.println("Floor: " + currentFloor);}  
  }*   
    
  **Code**  
      public void goUp() {  
              System.out.println("Going up one floor.");  
              currentFloor++;  
              System.out.println("Floor: " + currentFloor);  
          }  
    
          public void goDown() {  
    
              if (currentFloor == MIN\_FLOORS) {  
                  System.out.println("Cannot Go down");  
              }  
              else {  
                  System.out.println("Going down one floor.");  
                  currentFloor--;  
                  System.out.println("Floor: " + currentFloor);}  
              }  
          }  
     }

You can chain if and else constructs together to state multiple outcomes for several different  
expressions. This is the syntax.

Code

if (*boolean\_expression*) {  
    *code\_block1*  
} // end of if construct  
else if (*boolean\_expression*){  
    *code\_block2*  
} // end of else if construct  
else {  
    *code\_block3*  
}  
// program continues here

***boolean\_expression***

*The boolean\_expression is a combination of relational operators, conditional operators, and values resulting in a value of true or false.*

***code\_block1***

*The code\_block1 represents the lines of code that are executed if the expression is true.*

***code\_block2***

*The code\_block2 represents the lines of code that are executed if the expression is false and the condition in the second if is true.*

***code\_block3***

*The code\_block3 represents the lines of code that are executed if the expression in the second if also evaluates to false.*

This example is an IfElseDate class containing several chained if/else constructs that determine how many days there are in a month.

Code

...  
public void calculateNumDays() {  
  
        if (month == 1 || month == 3 || month == 5 || month == 7 ||  
            month == 8 || month == 10 || month == 12) {  
            System.out.println("There are 31 days in that month.");  
        }  
  
        else if (month == 2) {  
            System.out.println("There are 28 days in that month.");  
        }  
            else if (month == 4 || month == 6 || month == 9 || month == 11) {  
                System.out.println("There are 30 days in that month.");  
            }  
  
        else {  
            System.out.println("Invalid month.");  
...

Note

*Although this code is syntactically correct, chaining if/else statements can result in confusing code and should be avoided.*

The calculateNumDays method chains three if/else statements together to determine the number of days in a month:

Code

...  
public void calculateNumDays() {  
  
        if (month == 1 || month == 3 || month == 5 || month == 7 ||  
            month == 8 || month == 10 || month == 12) {  
            System.out.println("There are 31 days in that month.");  
        }  
  
        else if (month == 2) {  
            System.out.println("There are 28 days in that month.");  
        }  
            else if (month == 4 || month == 6 || month == 9 || month == 11) {  
                System.out.println("There are 30 days in that month.");  
            }  
  
        else {  
            System.out.println("Invalid month.");  
...

* When the first if statement is true, the first code block executes.   
  *The relevant code is:  
    
  if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 || month == 12) {  
  System.out.println("There are 31 days in that month.");  
  }*   
    
  **Code**  
  ...  
  public void calculateNumDays() {  
    
          if (month == 1 || month == 3 || month == 5 || month == 7 ||  
      month == 8 || month == 10 || month == 12) {  
              System.out.println("There are 31 days in that month.");  
          }  
    
          else if (month == 2) {  
              System.out.println("There are 28 days in that month.");  
          }  
              else if (month == 4 || month == 6 || month == 9 || month == 11) {  
                  System.out.println("There are 30 days in that month.");  
              }  
    
          else {  
              System.out.println("Invalid month.");  
  ...
* The second code block executes when the first if  
  statement is false and the first else statement is true.   
  *The relevant code is:  
    
  else if (month == 2) {  
  System.out.println("There are 28 days in that month.");  
  }*   
    
  **Code**  
  ...  
  public void calculateNumDays() {  
    
          if (month == 1 || month == 3 || month == 5 || month == 7 ||  
      month == 8 || month == 10 || month == 12) {  
              System.out.println("There are 31 days in that month.");  
          }  
    
          else if (month == 2) {  
              System.out.println("There are 28 days in that month.");  
          }  
              else if (month == 4 || month == 6 || month == 9 || month == 11) {  
                  System.out.println("There are 30 days in that month.");  
              }  
    
          else {  
              System.out.println("Invalid month.");  
  ...
* The third code block executes when the first if  
  statement is false, the first else statement is false, and the second else statement is true.   
  *The relevant code is:  
    
  else if (month == 4 || month == 6 || month == 9 || month == 11) {  
  System.out.println("There are 30 days in that month.");  
  }*   
    
  **Code**  
  ...  
  public void calculateNumDays() {  
    
          if (month == 1 || month == 3 || month == 5 || month == 7 ||  
      month == 8 || month == 10 || month == 12) {  
              System.out.println("There are 31 days in that month.");  
          }  
    
          else if (month == 2) {  
              System.out.println("There are 28 days in that month.");  
          }  
              else if (month == 4 || month == 6 || month == 9 || month == 11) {  
                  System.out.println("There are 30 days in that month.");  
              }  
    
          else {  
              System.out.println("Invalid month.");  
  ...
* The final code block executes when all the prior statements are false.   
  *The relevant code is:  
    
  else {  
  System.out.println("Invalid month.");*   
    
  **Code**  
  ...  
  public void calculateNumDays() {  
    
          if (month == 1 || month == 3 || month == 5 || month == 7 ||  
      month == 8 || month == 10 || month == 12) {  
              System.out.println("There are 31 days in that month.");  
          }  
    
          else if (month == 2) {  
              System.out.println("There are 28 days in that month.");  
          }  
              else if (month == 4 || month == 6 || month == 9 || month == 11) {  
                  System.out.println("There are 30 days in that month.");  
              }  
    
          else {  
              System.out.println("Invalid month.");  
  ...

**2. Working with an if/else statement**

You'll often create classes that use if and if/else constructs. For instance, you could create the DateTwo class that uses if/else statements to display the day of the week based on the value of a variable.  
  
To do this, you need to code the class to evaluate a numeric field in order to determine the day of the week that corresponds to that number.

You create a new project from existing sources, remembering to set the Source/Binary Format property.  
  
Open the new project and using the New File wizard, create a new Java class called DateTwo. Declare and initialize a member field for this class called dayNumber. The value assigned should be a number between 1 and 7, where the number 1 represents Monday as the beginning of the week and 7 represents Sunday as the end of the week.

Code

public int dayNumber = 1;

Note

*Be sure to use the int data type.*

You now want to create a displayDay method in the DateTwo class. You can use an if/else construct to inspect the value of dayNumber, and display an error message if an invalid number is found.  
  
You can use pseudocode to write the body of the displayDay method. Each if condition should check the value of dayNumber. You can use the increment sign to do this. Within the if blocks, print out the day of the week.

Code

if (condition1) {  
    // print corresponding day  
}else if (condition2) {  
    // print corresponding day  
}else if (condition3)  
    …  
}else {  
    // if none of the conditions is true  
}

The value of dayNumber is inspected in the first step. If dayNumber does not equal a number between 1 and 7, the second step takes place, which is to print out an error message. This will be in the final else block. This is the resulting code.

Code

public void displayDay() {  
        if (dayNumber == 1) {  
            System.out.println("Monday");  
        } else if (dayNumber == 2) {  
            System.out.println("Tuesday");  
        } else if (dayNumber == 3) {  
            System.out.println("Wednesday");  
        } else if (dayNumber == 4) {  
            System.out.println("Thursday");  
        } else if (dayNumber == 5) {  
            System.out.println("Friday");  
        } else if (dayNumber == 6) {  
            System.out.println("Saturday");  
        } else if (dayNumber == 7) {  
            System.out.println("Sunday");  
        } else {  
            System.out.println(dayNumber + " entered.  This is not a valid date.");  
        }  
   }

You can then save and compile your class and then execute it by running the DateTwoTest class. You can then check the output in the Output window.  
  
To test the class, you should repeat this step several times by assigning different values to the DateTwo member field.

You also want to write a class called Clock that uses if/else statements to display the part of day depending upon the time of day.

Graphic

*A table displays with two columns: Time of day and Part of day. For 8:01 to 12:00, the part of the day is Morning. 12:01 to 17:00 is Afternoon, 17:01 to 24:00 is Evening, and 0:01 to 08:00 is Early Morning.*

You create a new Java class called Clock that contains an int field called currentTime. Initialize this field to hold the hour of the day. For example, 400 can represent 04:00, and 1505 can represent 15:05.

Code

public int currentTime = 2100;

In the Clock class, create a displayPartOfDay method to display the part of the day associated with the value of the currentTime field. For instance, if the currentTime field equals 2100, you would display Evening. You need not check for values outside the range of 1 to 2400.

Code

public void displayPartOfDay(){  
        if (currentTime >= 801 && currentTime <= 1200){  
            System.out.println ("Morning");  
        }else if (currentTime >= 1201 && currentTime <= 1700){  
            System.out.println ("Afternoon");  
        }else if (currentTime >= 1701 && currentTime <= 2400){  
            System.out.println ("Evening");  
        }else {  
            System.out.println ("Early morning");  
        }// end if/else

Save and compile your class and execute it by running the ClockTest class. You should repeat this step several times by assigning different values to the Clock member variable.  
  
Remember that a leading zero indicates an octal value, so the program does not compile if you set currentTime to 0800. You need to make sure you specify currentTime as 800 for 8:00 AM to successfully compile the program.

Question

You are comparing two values – guess and number. You want to print "Equal" if the value of guess equals the value of number and "Not Equal" if the values are different. Which statement achieves this?

**Options:**

1. if (guess == number)  
      System.out.println("Equal");  
   else  
      System.out.println("Not equal");
2. if (guess == number)  
      System.out.println("Equal");  
      System.out.println("Not equal");
3. if (guess == number)  
      System.out.println("Not equal");  
   else if (guess != number)  
      System.out.println("Equal");

Answer

***Option 1:****Correct. By using the if/else construct, you are able to choose between two statements depending on the conditional check.*

***Option 2:****Incorrect. The statement is missing the else keyword and results in both "Equal" and "Not equal" being printed onscreen if the values are equal. "Not equal" is printed onscreen if the values are not equal.*

***Option 3:****Incorrect. The statement reverses the intended results and prints "Not equal" if the values are equal, and "Equal" if the values are not equal.*

**Correct answer(s):**

1. if (guess == number)  
   System.out.println("Equal");  
else  
   System.out.println("Not equal");

**3. Using switch statements**

Another keyword used in decision making is the switch keyword. The switch construct helps you avoid confusing code because it simplifies the organization of the various branches of code that can be executed.

Syntax

switch (*variable*) {  
    case *literal\_value*:  
*code\_block*        [break;]  
    case *another\_literal\_value*:  
*code\_block*        [break;]  
    [default:]  
*code\_block*}

The IfElseDate class example could be rewritten by using a switch construct. This is the syntax for the switch construct.

Syntax

switch (*variable*) {  
    case *literal\_value*:  
        *code\_block*  
        [break;]  
    case *another\_literal\_value*:  
        *code\_block*  
        [break;]  
    [default:]  
        *code\_block*  
}

*The switch keyword indicates a switch statement.*

*The variable is the variable whose value you want to test. The variable can be only of type String, char, byte, short, or int.*

*The case keyword indicates a value that you are testing. The combination of the case keyword and a literal\_value is referred to as a case label.*

*The literal\_value is any valid value that a variable might contain. You can have a case label for each value that you want to test. Literal values cannot be variables, expressions, String, or method calls. Literal values can be constants – final variables such as MAX\_NUMBER defined somewhere else– or literals such as A or 10.*

*The [break;] statement is an optional keyword that causes the flow of code to immediately exit the switch statement. Without a break statement, all code\_block statements following the accepted case statement are executed – until a break statement or the end of the switch construct is reached.*

In this example, the SwitchDate class uses a switch construct to determine how many days there are in a month. The calculateNumDays method in the SwitchDate class uses a switch statement to branch on the value of the month variable. If the month variable is equal to 1, 3, 5, 7, 8, 10, or 12, the code jumps to the appropriate case label and then drops down to execute the relevant code.

Graphic

*The relevant code is:  
  
System.out.println(“There are 31 days in that month.”).*

Code

    public class SwitchDate {  
  
        public int month = 10;  
  
        public void calculateNumDays() {  
  
            switch(month) {  
            case 1:  
                case 3:  
                case 5:  
                case 7:  
                case 8:  
                case 10:  
                case 12:  
                    System.out.println("There are 31 days in that month.");  
                    break;  
                default:  
                    System.out.println("Invalid month.");  
                    break;  
...

Supplement

*Selecting the link title opens the resource in a new browser window.*

**Job aid**

Access the job aid [SwitchDate class](javascript:doWindow('./ja_jsef_a03_it_enust204_frame.html')) to review the source code for the class.

You can use switch constructs for

* equality tests
* tests against a single value, such as customerStatus
* tests against the value of an int, short, byte, or char type and Strings, and
* tests against a fixed value known at compile time

Suppose you create a class called Month that uses switch statements to display the name of the month based upon the numeric value of a field.

You can do this by following these steps:

* Begin by creating a new Java class called Month.
* Declare an int field called monthNumber in the Month class. Assign a value to the field that is between 1 and 12, where 1 represents the month of January and 12 represents the month of December.   
    
  **Code**  
  public int monthNumber = 6;
* Create a new method called displayMonth in the Month class. This method uses a switch construct to inspect the value of the monthNumber field and display the corresponding name of the month. The displayMonth method should also display an error message if an invalid number is used.   
    
  **Code**  
  public void displayMonth() {  
          switch (monthNumber) {  
              case 1:  
                  System.out.println("January");  
                  break;  
              case 2:  
                  System.out.println("Febrary");  
                  break;  
              case 3:  
                  System.out.println("March");  
                  break;  
              case 4:  
                  System.out.println("April");  
                  break;  
              ...  
              case 12:  
                  System.out.println("December");  
                  break;  
              default:  
                  System.out.println(monthNumber + " is not a valid number.");  
                  break;  
          }// end of switch  
      } // end of displayMonth method
* Save and compile your program and execute it by running the MonthTest class. Repeat this step several times by assigning different values to the monthName field to test the class.

Question

You are writing an application that utilizes the integer value stored in the variable numDay to display the day of the week. You wish to use a switch construct to compare the values and set the day.  
  
Complete the code to do this.

**Code**  
public class Days {  
public Days() {}  
  public void printWeekday(int numDay) {  
    String day;  
    INSERT THE MISSING CODE {  
  }  
}

**Options:**

1. switch (numDay)
2. switch (day)
3. switch ( )
4. switch (int)

Answer

***Option 1:****Correct. The variable value is the variable whose value you want to test. The variable value can only be type char, byte, short, or int. In this case, you specify the integer value numDay.*

***Option 2:****Incorrect. You need to specify the variable whose value you want to test, which in this case is numDay.*

***Option 3:****Incorrect. You need to specify the switch keyword, followed by the variable you want to test enclosed in parentheses. In this case, the variable is numDay.*

***Option 4:****Incorrect. You specify the variable in the parentheses after the switch keyword, not its type.*

**Correct answer(s):**

1. switch (numDay)

**4. Summary**

In this topic, you've learned how to use if/else constructs and switch statements in Java programming.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#top)

Creating and Using Arrays

Learning Objectives

*After completing this topic, you should be able to*

* *create a one-dimensional array*
* *work with two-dimensional arrays*

**1. One-dimensional arrays**

Consider a program where you store the ages of ten people. You could create individual variables to hold each of the ten values. You could do this using this code, but there are problems with this approach. What if you had to store 1,000 ages or 10,000 ages? As the number of values increases, your program becomes increasingly unmanageable. What if you had to find the average age, or sort the ages into ascending order? You’d have to refer to each variable individually in your code.  
  
Arrays in Java – and related constructs, such as Lists – provide a much more convenient way to work with sets of data.

Code

int ageOne = 27;  
int ageTwo = 12;  
int ageThree = 82;  
int ageFour = 70;  
int ageFive = 54;  
int ageSix = 6;  
int ageSeven = 1;  
int ageEight = 30;  
int ageNine = 34;  
int ageTen = 42;

The Java programming language allows you to group multiple values of the same type, or lists, using arrays. Arrays are useful when you have related pieces of data, such as the ages of several people, but you do not want to create separate variables to hold each piece of data.

You can create an array of primitive types, such as int, or an array of references to object types, such as Shirt or String. Each part of the array is an element. If you declare an array of 100 int types, there are 100 elements. You can access each specific element within the array using its location or index in the array.

Graphic

*In this example, there are arrays for int types, Shirt types, and String types.*

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.  
  
Each item in an array is called an element, and each element is accessed by its numerical index. Numbering begins with zero. The sixth element, for example, would therefore be accessed at index 5.

Graphic

*In this example, there is an array named ages with eight elements. The elements are as follows: 27, 12, 82, 70, 54, 1, 30, and 34. The indices are numbered as 0 through 7, with the first index being 0. The element at index 5 is 1.*

The length of an array can be accessed using dot notation to access the length field. Assuming the array is called ages, you can use the given code to assign a value of 8 to int agesLength.

Code

int agesLength = ages.length;

Arrays are handled by an implicit Array object, which isn't available in the Java API. Just like with any object, you must declare an object reference to the array, instantiate an Array object, and then initialize the Array object before you can use it.

This is the syntax used to declare a one-dimensional array.

Syntax

*type [] array\_identifier*;

*The type represents the primitive data type or object type for the values stored in the array.*

*The [] informs the compiler that you are declaring an array.*

*The array\_identifier is the name that you are assigning to refer to the array.*

You can do the following:

* declare arrays of types char and int, and   
    
  **Code**  
  char [] status;  
  int [] ages;
* declare arrays of object references of types Shirt and String   
    
  **Code**  
  Shirt [] shirts;  
  String [] names;

When you declare an array, the compiler and the Java Virtual Machine, or JVM, have no idea how large the arrays will be because you have declared a reference variable that does not currently point to any objects.

Before you can initialize an array, you must instantiate an Array object large enough to hold all of the values in the array. Instantiate an array by defining the number of elements in the array. You can use this syntax.

Syntax

*array\_identifier* = *new* *type* [*length*];

*The array\_identifier is the name you are assigning to reference the array.*

*The type represents the primitive data type or object type for the values stored in the array.*

*The length represents the size, in number of elements, of the array.*

When you instantiate an Array object, every primitive element is initialized to the zero value for the type you specified. In the case of the char array called status, each value is initialized to \u0000, the null character of the Unicode character set. For the int array called ages, the initial value is the integer value 0. For the names and shirt arrays, the object references are initialized to null.

Code

status = new char [20];  
ages = new int [5];  
names = new String [7];  
shirts = new Shirt [3];

You can fill the contents of an array after you have created it using this syntax.

Code

ages[0] = 19;  
ages[1] = 42;  
ages[2] = 92;  
ages[3] = 33;

Syntax

*array\_identifier*[*index*] = value;

*The array\_identifier is the name you are assigning to the array.*

*The index represents the location in the array where the value will be placed.*

In this sample code, the new keyword is used to create the Shirt objects and to place the references to the Shirt objects into each position in the array.  
  
The index to the first element of an array is 0 and the index to the last element of the array is the length of the array minus 1. For example, the last element of a six-element array is index 5.

Code

shirts[0] = new Shirt();  
shirts[1] = new Shirt();  
shirts[2] = new Shirt();

If you know the values you want in your array at the time that you declare it, you can declare, instantiate, and set the values for an Array object in the same line of code.

Syntax

*type* [] *array\_identifier* = {*values*};

*The type represents the primitive data type or object type for the values to be stored.*

*The angle brackets informs the compiler that you are declaring an array.*

*The array\_identifier is the name you are assigning to the array.*

*The {values} represents a list of values or expressions that you want to store in the array, separated by commas.*

These statements combine declaration, instantiation, and initialization. The new keyword is used to instantiate the Shirt object so that a reference to that object can be placed in the array.  
  
However, the final example will return an error. You can't declare and initialize an array in separate lines by using the comma-separated list technique.

Graphic

*The final example code is:  
  
int [] ages;  
ages = {19, 42, 92, 33, 46};*

Code

int [] ages = {19, 42, 92, 33, 46};  
  
Shirt [] shirts = {new Shirt(), new Shirt(), new Shirt()};  
  
int [] ages;  
ages = {19, 42, 92, 33, 46};

Each element of an array is accessed using its index. To access a value from the array, state the  
array name and the index number for the element in square brackets on the right side of an assignment operator.  
  
You can set a value using this code.

Code

status[0] = '3';  
names[1] = "Fred Smith";  
ages[1] = 19;  
prices[2] = 9.99F;

You can get a value using this code.

Code

char s = status[0];  
String name = names [1];  
int age = ages[1];  
double price = prices[2];

Arrays are objects referred to by an object reference variable. The way a primitive array is stored in memory differs from the way a primitive data type is stored in memory.  
  
The value of the size variable, which is a char primitive, is L. The value of sizes[] is 0x334009, and it points to an object of type array of char types with three values. The value of sizes[0] is char S, the value of sizes[1] is char M, and the value of sizes[2] is char L.

Code

char size = 'L'  
char [] sizes = {'S', 'M', 'L'};

An object reference array is stored in memory. The value of the myShirt object reference is x034009, which is an address to an object of type Shirt with the values 0, 0.0, and U.

The value of the shirts[ ] object reference is x99f311, which is an address to an object of type array – of Shirt object references – containing three object references.

**0x00099**

The value of the shirts[0] index is 0x00099, which is an object reference pointing to an object  
of type Shirt.

**0x00327**

The value of the shirts[1] index is 0x00327, which is an object reference pointing to another  
object of type Shirt.

**0x00990**

The value of the shirts[2] index is 0x00990, which is an object reference pointing to another object of type Shirt.

**2. Working with one-dimensional arrays**

You need to create an array containing the number of vacation days that an employee at the Duke's Choice company receives, based upon the number of years that the employee has worked for Duke's Choice.

Graphic

*A table displays with two columns: Years of employment and Vacation days. Employees with up to 1 year of employment get 10 days vacation; those with 1, 2, or 3 years get 15 days vacation; those with 4 or 5 years get 20 days vacation; and those with 6 or more years get 25 days vacation.*

You start by creating a new Java class called VacationScale. You need to declare, but not initialize, two public fields to this class as follows:

Code

    public int[] vacationDays;  
    public int yearsOfService;

* int array called vacationDays, and
* int called yearsOfService

Note

*Use the square brackets next to the data type to indicate that this field is an array.*

Then create a method in the VacationScale class called setVacationScale by

Code

    public void setVacationScale() {  
        vacationDays = new int[7];  
        vacationDays[0] = 10;  
        vacationDays[1] = 15;  
        vacationDays[2] = 15;  
        vacationDays[3] = 15;  
        vacationDays[4] = 20;  
        vacationDays[5] = 20;  
        vacationDays[6] = 25;  
    }

**initializing the vacationDays array, and**

Use the new keyword to initialize the vacationDays array. Supply the size of the array within the square brackets.

**assigning each array element**

Assign each array element, beginning with vacationDays[0] with the appropriate number of days of vacation from the table. For example, an employee with 0 years of service is entitled to 10 vacation days. An employee with 1 year of service is entitled to 15 days of vacation.

The next step is to create a public method called displayVacationDays that displays the number of vacation days due to an employee with the years of service indicated in the yearsOfService field.  
  
You can use an if/else construct to check for an invalid yearsOfService – a negative number – and display an error message in this case.

Code

public void displayVacationDays(){  
    if(yearsOfService >= 0){   
        System.out.println("Vacation days: " +  
            vacationDays[yearsOfService]);  
    }else {  
        System.out.println("Invalid years of service");  
  
    }  
}

Note

*You can use a variable within the square brackets to represent the array index number.*

Next you save and compile your program and run the VacationScaleTest class to test your program. However, as it's currently written, the program throws an exception, or an error.  
  
The exception thrown by the Java Virtual Machine, or JVM, is an ArrayIndexOutOfBounds exception.

Graphic

*The output reads as follows:  
  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10  
at VacationScale.displayVacationDays (VacationScale.java:28)  
at VacationScaleTest.main(VacationScaleTest.java:15)  
Java result: 1*

The exception is thrown when an attempt has been made to access a non-existent index of an array. The index number that caused the exception, which is shown in the error message, is index #10. Remember that this array has 7 elements, indexed by numbers 0 through 6.

Why did the program try to access index 10? If you look at the displayVacationDays method, you will see that the yearsOfService field is used as the array index – as an argument to the System.out.println method.  
  
It is, of course, conceivable that an employee would have more than six years of service, which would be the highest index number in the array. The displayVacationDays method needs to be modified to account for more than six years of service.

Code

public void displayVacationDays(){  
    if(yearsOfService >= 0){   
        System.out.println("Vacation days: " +  
            vacationDays[yearsOfService]);  
    } else {  
        System.out.println("Invalid years of service");  
  
    }  
}

To modify the displayVacationDays method to account for more than six years of service, you can change the if/else construct to also check for a yearsOfService value that is >=6. All years of service greater than or equal to six now receive the same number of vacation days.  
  
For any yearsOfService value between zero and five, you can display the value of the array whose index corresponds to that value. For a yearsOfService of six and above, use the value referenced by the last array index.

Code

if(yearsOfService >= 0 && yearsOfService < 6){  
    System.out.println ("Days of vacation: " + vacationDays[yearsOfService]);  
} else if(yearsOfService >= 6){  
    System.out.println ("Days of vacation: " + vacationDays[6]);  
} else {  
    System.out.println ("Invalid years of service");  
}

Finally, you can save and compile the program and then test it again by running the VacationScaleTest class. All three of the test values for yearsOfService should be displayed in the output window.

Graphic

*The output reads as follows:  
  
run:  
Days of vacation: 15  
Days of vacation: 20  
Days of vacation: 25  
BUILD SUCCESSFUL (total time: 0 seconds)*

Question

Which statements accurately describe an array?

**Options:**

1. You can fill an array after you’ve created it
2. The index for the first element of an array is 1
3. The {} symbols inform the compiler you are declaring an array
4. You can declare, instantiate, and set the values for an Array object in the same line of code

Answer

***Option 1:****Correct. You can fill the contents of an array after you have created it. The syntax for setting the values in  
an array is array\_identifier[index] = value;.*

***Option 2:****Incorrect. The index to the first element of an array is zero and the index to the last element of the array is the length of the array minus one.*

***Option 3:****Incorrect. A set of square brackets informs the compiler that you are declaring an array.*

***Option 4:****Correct. If you know the values you want in your array at the time that you declare it, you can declare, instantiate, and set the values for an Array object in the same line of code. The syntax for this is type [ ] array\_identifier ={values};.*

**Correct answer(s):**

1. You can fill an array after you’ve created it  
4. You can declare, instantiate, and set the values for an Array object in the same line of code

**3. Two-dimensional arrays**

When you pass strings to your program on the command line, the strings are put in the args array. To use these strings, you must extract them from the args array and, optionally, convert them to their proper type – because the args array is of type String.  
  
In this example, the ArgsTest class extracts two String arguments passed on the command line and displays them.

Code

> java ArgsTest Hello World!  
args[0] is Hello  
args[1] is World!

The parameters are displayed as follows:

Code

> java ArgsTest Hello World!  
args[0] is Hello  
args[1] is World!

* the first parameter goes into args[0], and   
  *The relevant code is: args[0] is Hello*   
    
  **Code**  
  > java ArgsTest Hello World!  
  args[0] is Hello  
  args[1] is World!
* the second parameter goes into args[1]   
  *The relevant code is: args[1] is World!*   
    
  **Code**  
  > java ArgsTest Hello World!  
  args[0] is Hello  
  args[1] is World!

To add parameters on the command line, you must leave one or more spaces after the class name – in this case, ArgsTest – and one or more spaces between each parameter added.

Code

> java ArgsTest Hello World!  
args[0] is Hello  
args[1] is World!

NetBeans does not allow you to run a Java class from the command line, but you can set  
command-line arguments as a property of the project your code is in.

Code

public class ArgsTest {  
    public static void main (String args[]) {  
        System.out.println("args[0] is " + args[0]);  
        System.out.println("args[1] is " + args[1]);  
    }  
}

Numbers can be typed as parameters.

Code

> java ArgsTest 2 3  
Total is: 23  
Total is: 5

The main method treats everything you type as a literal string. If you want to use the string representation of a number in an expression, you must convert the string to its numerical equivalent. Every data type has an associated class containing static utility methods for converting strings to that data type – such as Integer class for int, Byte class for byte, and Long class for long.   
  
For example, to convert the first argument passed to the main method to an int type, you can use the highlighted code.

Graphic

*The highlighted code is: Integer.parseInt(args[])*

Code

    public class ArgsTest {  
        public static void main (String args[]) {  
            System.out.println("Total is: " + (args[0] + args[1]));  
  
            int arg1 = Integer.parseInt(args[0]);  
            int arg2 = Integer.parseInt(args[1]);  
            System.out.println("Total is: " + (arg1 + arg2));  
        }  
   }

Note

*The parentheses around arg1 + arg2 are required so that the + sign indicates addition rather than concatenation.*

You also can store matrices of data using multidimensional arrays – arrays of arrays, of arrays, and so on. A two-dimensional array, or an array of arrays, is similar to a spreadsheet with multiple columns and multiple rows. Each column represents one array or list of items.  
  
In this example, a two-dimensional array is created of two arrays. The descriptive names Week 1, Week 2, Monday, Tuesday, and so on would not be used to access the elements of the array. Instead, Week 1 would be index 0 and Week 4 would be index 3 along that dimension, while Sunday would be index 0 and Saturday would be index 6 along the other dimension.

Graphic

*In this table, the columns are Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday. The rows are Week 1, Week 2, Week 3, and Week 4.*

Two-dimensional arrays require an additional set of square brackets. The process of creating and using two-dimensional arrays is otherwise the same as with one-dimensional arrays. This is the syntax.

Syntax

*type* [ ] [ ] *array\_identifier*;

*The type represents the primitive data type or object type for the values stored in the array.*

*The two sets of angle brackets inform the compiler that you are declaring a two-dimensional array.*

*The array\_identifier is the name you have assigned the array during declaration.*

This example declares an array of arrays for sales amounts.

Code

int [][] yearlySales;

This is the syntax for instantiating a two-dimensional array.

Syntax

*array\_identifier* = new type [*number\_of\_arrays*] [*length]*;

*The array\_identifier is the name you have assigned the array during declaration.*

*The number\_of\_arrays is the number of arrays within the array.*

*The length is the length of each array within the array.*

This example instantiates an array of arrays for quarterly sales amounts over five years.  
  
The yearlySales array contains five elements of the type int array, which form five sub-arrays. Each sub-array is four elements in size and tracks the sales for one year over four quarters.

Graphic

*In this table, the columns are Quarter 1, Quarter 2, Quarter 3, and Quarter 4. The rows are Year 1, Year 2, Year 3, Year 4, and Year 5.*

Code

// Instantiates a 2D array: 5 arrays of 4 elements each  
yearlySales = new int[5][4];

When setting or getting values in a two-dimensional array, you need to indicate the index number in the array by using a number to represent the row, followed by a number to represent the column. This example contains five assignments of values to elements of the yearlySales array.

Code

yearlySales[0][0] = 1000;  
yearlySales[0][1] = 1500;  
yearlySales[0][2] = 1800;  
yearlySales[1][0] = 1000;  
yearlySales[3][3] = 2000;

The newly set values are contained in the appropriate arrays.

Graphic

*Year 1, Quarter 1, contains the value 1000. Year 1, Quarter 2, contains the value 1500. Year 1, Quarter 3, contains the value 1800. Year 2, Quarter 1, contains the value 1000. Finally, Year 4, Quarter 4, contains the value 2000.*

Question

Given a two-dimensional array that contains 5 arrays of 4 elements each, which code sample correctly sets the value of the fourth row and column to 1000?

**Options:**

1. yearlySales[0] [0] = 1000;
2. yearlySales[4] [4] = 1000;
3. yearlySales[3] [3] = 1000;
4. yearlySales[2] [3] = 1000;

Answer

***Option 1:****Incorrect. This code sample would set the value of the first row and first column to 1000.*

***Option 2:****Incorrect. This code sample would set the value of the fifth row and fifth column – if they existed in the two-dimensional array – to 1000.*

***Option 3:****Correct. The index to the first element of an array is 0, so this code sample would set the value of the fourth column and fourth row to 1000.*

***Option 4:****Incorrect. This code sample would set the value of the third column and fourth row to 1000.*

**Correct answer(s):**

3. yearlySales[3] [3] = 1000;

**4. Summary**

In this topic, you've learned how to create and work with one-dimensional and two-dimensional arrays in Java programming.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#top)

Working with Java Arrays

Learning Objective

*After completing this topic, you should be able to*

* *work with ArrayList*

**1. Working with ArrayList**

Arrays are not the only way to store lists of related data. You can use ArrayList to do this, which is one of a number of list classes. It can store only objects, not primitives. It has a set of useful methods for managing its elements, including add(), get(), remove(), indexOf() and many others.  
  
You do not need to specify a size when you instantiate an ArrayList. As you add more elements, the ArrayList will grow as necessary. You can specify an initial capacity, but it is not mandatory to do so.

For lists that are very dynamic, it may be easier to work with a specialized List type object. This can free you from having to write code to

* keep track of the index of the last piece of data added
* keep track of how full the array is and determine if it needs to be resized, and
* increase the size of the array by creating a new one and copying the elements from the current one into it

The ArrayList class is in the package java.util. To refer to ArrayList in your code, you can fully qualify the statement shown.

Code

java.util.ArrayList myList;

Alternatively, you can add the import statement at the top of the class.

Code

import java.util.ArrayList;  
public class ArrayListExample {  
    public static void main (String args[]) {  
        ArrayList myList;  
    }  
}

Classes in the Java programming language are grouped into packages depending on their functionality.   
  
For example, all classes related to the core Java programming language are in the java.lang package, which contains classes that are fundamental to the Java programming language, such as String, Math, and Integer. Classes in the java.lang package can be referred to in code by just their class name. They do not require full qualification or the use of an import statement.

All classes in other packages, such as ArrayList, require that you fully qualify them in the code, or that you use an import statement so that they can be referred to directly in the code.

Code

import java.util.ArrayList;  
public class ArrayListExample {  
    public static void main (String args[]) {  
        ArrayList myList;  
    }  
}

The import statement can be used for the following:

* for just the class in question, or   
    
  **Code**  
  java.util.ArrayList;
* for all classes in the package   
    
  **Code**  
  java.util.\*;

Declaring an ArrayList is exactly the same as declaring any other reference type.

Graphic

*The code is:  
  
ArrayList myList;*

Code

ArrayList myList;  
  
myList = new ArrayList();  
  
myList.add("John");  
myList.add("Ming");  
myList.add("Mary");  
myList.add("Prashant");  
myList.add("Desmond");  
  
myList.remove(0);  
myList.remove(myList.size()-1);  
myList.remove("Mary");

Likewise, instantiating an ArrayList is the same as instantiating any other object. You can check the documentation for other possibilities for instantiating.

Graphic

*The code is:  
  
myList = new ArrayList();*

Code

ArrayList myList;  
  
myList = new ArrayList();  
  
myList.add("John");  
myList.add("Ming");  
myList.add("Mary");  
myList.add("Prashant");  
myList.add("Desmond");  
  
myList.remove(0);  
myList.remove(myList.size()-1);  
myList.remove("Mary");

There are a number of methods to add data to the ArrayList. In this example, the add() method is used to add a string. Each call to add adds a new element to the end of the ArrayList.

Graphic

*The relevant code is:  
  
myList.add("John");  
myList.add("Ming");  
myList.add("Mary");  
myList.add("Prashant");  
myList.add("Desmond");*

Code

ArrayList myList;  
  
myList = new ArrayList();  
  
myList.add("John");  
myList.add("Ming");  
myList.add("Mary");  
myList.add("Prashant");  
myList.add("Desmond");  
  
myList.remove(0);  
myList.remove(myList.size()-1);  
myList.remove("Mary");

Finally, a big advantage of ArrayList over an array is that there are many methods available for manipulating the data.

Graphic

*This is the relevant code:  
  
myList.remove(0);  
myList.remove(myList.size()-1);  
myList.remove("Mary");*

Code

ArrayList myList;  
  
myList = new ArrayList();  
  
myList.add("John");  
myList.add("Ming");  
myList.add("Mary");  
myList.add("Prashant");  
myList.add("Desmond");  
  
myList.remove(0);  
myList.remove(myList.size()-1);  
myList.remove("Mary");

In this example, the remove method is used to modify data.

Code

ArrayList myList;  
  
myList = new ArrayList();  
  
myList.add("John");  
myList.add("Ming");  
myList.add("Mary");  
myList.add("Prashant");  
myList.add("Desmond");  
  
myList.remove(0);  
myList.remove(myList.size()-1);  
myList.remove("Mary");

**myList.remove(0);**

*This removes the first element – in this case, the first element is John.*

**myList.remove(myList.size()-1);**

*This removes the last element as myList.size() gives the number of elements of the array, so the last one is the size minus 1 – in this case, Desmond is removed.*

**myList.remove("Mary");**

*This removes a specific element. In this case, we have the convenience of referring not to where the element is in the ArrayList, but rather to what it is.*

You can pass an ArrayList to System.out.println() and the resulting output will be [Ming, Prashant].

Code

ArrayList myList;  
  
myList = new ArrayList();  
  
myList.add("John");  
myList.add("Ming");  
myList.add("Mary");  
myList.add("Prashant");  
myList.add("Desmond");  
  
myList.remove(0);  
myList.remove(myList.size()-1);  
myList.remove("Mary");

**2. Populating and manipulating ArrayLists**

You need to create the NamesList class and the NamesListTest class in order to experiment with populating and manipulating ArrayLists. To begin, you create the two classes and then add a method to the NamesList class to populate the list and display its contents. Then you will add a method to manipulate the values in the list.

To create and populate an ArrayList, you can start by creating a new Java main class called NamesListTest. In the New File wizard, select Java Main Class as the type of Java class, leaving the main method empty for the time being. You will add code later.  
  
Next you create a new Java class called NamesList.

Code

public class NamesList {

In the NamesList class, declare a public ArrayList field called theList. Do not instantiate the theList field.

Graphic

*The relevant code is:  
  
public ArrayList theList;*

Code

public class NamesList {  
    public ArrayList theList;

Note

*When you type the word ArrayList, the editor will indicate a warning in the margin of this line. It does not recognize the ArrayList class. You must import this class to make it visible to the compiler.*

Put your cursor over the warning icon in the margin to see the warning description. Press **Alt**+**Enter** to view and select from a list of hints to solve this problem. Select **Add import for java.util.ArrayList**.

Graphic

*The list of hints also includes Create class "ArrayList" in package and Create class "ArrayList" in NamesList.*

The import statement will be placed at the top of the NamesList class, above the class declaration.

Graphic

*This is the relevant code:  
  
import java.util.ArrayList;*

Code

import java.util.ArrayList;  
  
/\*  
 \* To change this template, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/  
  
/\*\*  
 \*  
 \* @a**uthor**Administrator  
 \*/  
public class **NamesList** (  
    public ArrayList theList;

Finally, you add a new method to the NamesList class called setList.

Code

    public void setList(){

Code the method using these steps:

Code

    public void setList(){  
        theList = new ArrayList();  
        theList.add("Joe Smith");  
        theList.add("Mary Palmer");  
        theList.add("Jose Gonzalez");  
        theList.add("Linda Baker");  
          
                    System.out.println("Names List: " + theList);  
                    System.out.println("Size of ArrayList is: " + theList.size());  
   }

* Use the new keyword to instantiate theList.   
  *The relevant code is:  
    
  theList = new ArrayList();*   
    
  **Code**  
      public void setList(){  
          theList = new ArrayList();  
          theList.add("Joe Smith");  
          theList.add("Mary Palmer");  
          theList.add("Jose Gonzalez");  
          theList.add("Linda Baker");  
            
                      System.out.println("Names List: " + theList);  
                      System.out.println("Size of ArrayList is: " + theList.size());  
     }
* Invoke the add method of the theList object. Pass a String value containing first\_name and last\_name, separated by a space.   
  *The relevant code is:  
    
  theList.add("Joe Smith");*   
    
  **Code**  
      public void setList(){  
          theList = new ArrayList();  
          theList.add("Joe Smith");  
          theList.add("Mary Palmer");  
          theList.add("Jose Gonzalez");  
          theList.add("Linda Baker");  
            
                      System.out.println("Names List: " + theList);  
                      System.out.println("Size of ArrayList is: " + theList.size());  
     }
* Repeat the second step three more times, using a different name in each method invocation.   
  *The relevant code is:  
    
  theList.add("Mary Palmer");  
  theList.add("Jose Gonzalez");  
  theList.add("Linda Baker");*   
    
  **Code**  
      public void setList(){  
          theList = new ArrayList();  
          theList.add("Joe Smith");  
          theList.add("Mary Palmer");  
          theList.add("Jose Gonzalez");  
          theList.add("Linda Baker");  
            
                      System.out.println("Names List: " + theList);  
                      System.out.println("Size of ArrayList is: " + theList.size());  
     }
* Use System.out.println to print out all of the names within the theList object. Use a suitable label and concatenate the theList object to it.   
  *The relevant code is:  
    
  System.out.println("Names List: " + theList);*   
    
  **Code**  
      public void setList(){  
          theList = new ArrayList();  
          theList.add("Joe Smith");  
          theList.add("Mary Palmer");  
          theList.add("Jose Gonzalez");  
          theList.add("Linda Baker");  
            
                      System.out.println("Names List: " + theList);  
                      System.out.println("Size of ArrayList is: " + theList.size());  
     }
* Use System.out.println to print out the size (number of elements) of the theList object. Use the size method of the theList object and concatenate a suitable label.   
  *The relevant code is:  
    
  System.out.println("Size of ArrayList is: " + theList.size());*   
    
  **Code**  
      public void setList(){  
          theList = new ArrayList();  
          theList.add("Joe Smith");  
          theList.add("Mary Palmer");  
          theList.add("Jose Gonzalez");  
          theList.add("Linda Baker");  
            
                      System.out.println("Names List: " + theList);  
                      System.out.println("Size of ArrayList is: " + theList.size());  
     }

Then click **Save** to compile the program and then open the NamesListTest class in the editor. Follow these steps:

Code

public static void main(String[] args) {  
        NamesList names = new NamesList();  
        names.setList();

* in the main method, instantiate a NamesList object called "names", using the new keyword, and   
  *This is the relevant code:  
    
  NamesList names = new NamesList();*   
    
  **Code**  
  public static void main(String[] args) {  
          NamesList names = new NamesList();  
          names.setList();
* invoke the setList method of the names object   
  *This is the relevant code:  
    
  names.setList();*   
    
  **Code**  
  public static void main(String[] args) {  
          NamesList names = new NamesList();  
          names.setList();

Finally, you save and compile your program, and complete and run the NamesListTest class to test the program. This is the output.

Graphic

*The output reads as follows:  
  
run:  
Names List: [Joe Smith, Mary Palmer, Jose Gonzalez, Linda Baker]  
Size of ArrayList is: 4  
BUILD SUCCESSFUL (total time: 0 seconds)*

Question

Which statements accurately describe the ArrayList class?

**Options:**

1. Its size does not need to be specified when instantiating an ArrayList
2. It can store only primitives, not objects
3. It is one of a number of list classes
4. It is part of the java.lang package

Answer

***Option 1:****Correct. You don't need to specify a size when you instantiate an ArrayList. As you add more elements, the ArrayList will grow as necessary.*

***Option 2:****Incorrect. The ArrayList class can store only objects and cannot store primitives.*

***Option 3:****Correct. The ArrayList class is one of a number of list classes. List type objects can free you from having to write code to keep track of the index of the last piece of data added.*

***Option 4:****Incorrect. The ArrayList class is part of the java.util package.*

**Correct answer(s):**

1. Its size does not need to be specified when instantiating an ArrayList  
3. It is one of a number of list classes

You add another new method to the NamesList class called manipulateList, and you can now use it to manipulate the ArrayList by following these steps:

Code

theList.remove ("Joe Smith");  
System.out.println("Names list after removing element: " + theList);  
System.out.println("Size of the ArrayList: " +  
theList.size());  
theList.add(1, "Joe Smith");  
System.out.println("Names list after adding element: " + theList);  
System.out.println("Size of the ArrayList: " +  
theList.size());

* Remove a name in the ArrayList using the remove method and passing the full name in double quotes. This method takes an Object as an argument. A String literal, such as the quote-enclosed full name, is an object.   
  *The relevant code is:  
    
  theList.remove ("Joe Smith");*   
    
  **Code**  
  theList.remove ("Joe Smith");  
  System.out.println("Names list after removing element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());  
  theList.add(1, "Joe Smith");  
  System.out.println("Names list after adding element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());
* Use System.out.println to print the theList object, using a label.   
  *The relevant code is:  
    
  System.out.println("Names list after removing element: "  
  + theList);*   
    
  **Code**  
  theList.remove ("Joe Smith");  
  System.out.println("Names list after removing element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());  
  theList.add(1, "Joe Smith");  
  System.out.println("Names list after adding element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());
* Use System.out.println to print the current size of the ArrayList, using an appropriate label.   
  *The relevant code is:  
    
  System.out.println("Size of the ArrayList: " +  
  theList.size());*   
    
  **Code**  
  theList.remove ("Joe Smith");  
  System.out.println("Names list after removing element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());  
  theList.add(1, "Joe Smith");  
  System.out.println("Names list after adding element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());
* Use the add method of the ArrayList to add the name you just removed back into the ArrayList, but at a different location.   
  *The relevant code is:  
    
  theList.add(1, "Joe Smith");*   
    
  **Code**  
  theList.remove ("Joe Smith");  
  System.out.println("Names list after removing element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());  
  theList.add(1, "Joe Smith");  
  System.out.println("Names list after adding element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());

You can then complete these steps:

Code

theList.remove ("Joe Smith");  
System.out.println("Names list after removing element: " + theList);  
System.out.println("Size of the ArrayList: " +  
theList.size());  
theList.add(1, "Joe Smith");  
System.out.println("Names list after adding element: " + theList);  
System.out.println("Size of the ArrayList: " +  
theList.size());

* Use a suitable label when printing the newly modified contents of the theList object.   
  *The relevant code is:  
    
  System.out.println("Names list after adding element: "  
  + theList);*   
    
  **Code**  
  theList.remove ("Joe Smith");  
  System.out.println("Names list after removing element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());  
  theList.add(1, "Joe Smith");  
  System.out.println("Names list after adding element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());
* Use a suitable label when printing the new size of the theList object.   
  *The relevant code is:  
    
  System.out.println("Size of the ArrayList: " +  
  theList.size());*   
    
  **Code**  
  theList.remove ("Joe Smith");  
  System.out.println("Names list after removing element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());  
  theList.add(1, "Joe Smith");  
  System.out.println("Names list after adding element: " + theList);  
  System.out.println("Size of the ArrayList: " +  
  theList.size());

Note that the add method is overloaded, meaning that it has two different method signatures. One of the add methods takes an Object and appends it to the end of the ArrayList.  
  
The other method takes an index number and an Object. It inserts the Object before the referenced index number, pushing all remaining list elements over one index number. It's recommended that you use this add method.

Code

theList.remove ("Joe Smith");  
System.out.println("Names list after removing element: " + theList);  
System.out.println("Size of the ArrayList: " +  
theList.size());  
theList.add(1, "Joe Smith");  
System.out.println("Names list after adding element: " + theList);  
System.out.println("Size of the ArrayList: " +  
theList.size());

In the main method of the NamesListTest class, you can now invoke the manipulateList method of the names object.

Code

public static void main(String[] args) {  
        NamesList names = new NamesList();  
        names.setList();  
        names.manipulateList();  
    }

Save and compile the program and run the NamesListTest class to test the program. The output depends upon the name you removed and added, and the index number you used in the add method.  
  
In the example, Joe Smith was previously located at index position 0 and Mary Palmer was at index position 1. After removing Joe Smith, Mary Palmer moved to index position 0 and Jose Gonzalez was at index position 1. Joe Smith was then added at index position 1, pushing Jose Gonzalez over to index position 2.

Graphic

*The output reads as follows:  
  
run:  
Names List: [Joe Smith, Mary Palmer, Jose Gonzalez, Linda Baker]  
Size of ArrayList is: 4  
Names list after removing element: [Mary Palmer, Jose Gonzalez, Linda Baker]  
Size of ArrayList: 3  
Names list after adding element: [Mary Palmer, Joe Smith, Jose Gonzalez, Linda Baker]  
Size of ArrayList: 4  
BUILD SUCCESSFUL (total time: 0 seconds)*

Question

You have an ArrayList daysList object that holds the values of the days of the week in order starting with Sunday. You want to remove the value Sunday. Which code segments can you use?

**Options:**

1. daysList.remove("Sunday");
2. daysList.remove(0);
3. daysList.remove(daysList.size()-1);
4. daysList.remove(6);

Answer

***Option 1:****Correct. By using the remove method and specifying the exact ArrayList value you want to remove, you can remove the value of Sunday from the ArrayList.*

***Option 2:****Correct. The first index value is 0, so removing value 0 removes the first value, which is Sunday.*

***Option 3:****Incorrect. This code sample would remove the final value in the ArrayList, which is Saturday.*

***Option 4:****Incorrect. This code sample would remove the seventh value in the ArrayList, which is Saturday.*

**Correct answer(s):**

1. daysList.remove("Sunday");  
2. daysList.remove(0);

You want to write a guessing game that accepts an argument and displays an associated message. You create a class that accepts a runtime argument between 1 and 5, inclusive. You also randomly generate a number between 1 and 5 in the class and compare the value of the argument with the randomly generated number.

Code

public static void main(String[] args) {  
     int randomNum = 0;  
        int guess;

Begin by creating a new Java Main Class called GuessingGame. You can then declare two int variables in the main method.

Code

int randomNum = 0;  
int guess;

Add code to the main method to accept a single argument of any number in the range of 1 to 5, inclusive, or the word "help." The high level steps are described in the pseudo code.

Code

if length of args array == 0 or value of args[0] = "help"  
    print a Correct Usage message  
else  
    randomNum = a generated random number 1 - 5   
    guess = integer value of args[0]  
  
    if argument < 1 or > 5  
        print an error message (invalid argument)  
    else  
        if argument == randomNum  
            print congratulations message  
        else  
            print a "Sorry; try again" message

Keep these points in mind when adding code to the main method:

Code

public static void main(String[] args) {  
     int randomNum = 0;  
        int guess;  
          
        if (args.length == 0 || args[0].compareTo("help") == 0){  
            System.out.println ("Usage: java GuessingGame [argument]");  
            System.out.println();  
            System.out.println ("Enter a number from 1 - 5 as your guess.");  
        }  
        else {  
            randomNum = ((int) (Math.random()\*5) + 1);  
            guess = Integer.parseInt(args[0]);

* Use the compareTo method of the String class to match the args[0] to "help." Elements of the args array will always be Strings.   
  *The relevant code is:  
    
  if (args.length == 0 || args[0].compareTo("help") == 0){*   
    
  **Code**  
  public static void main(String[] args) {  
       int randomNum = 0;  
          int guess;  
            
          if (args.length == 0 || args[0].compareTo("help") == 0){  
              System.out.println ("Usage: java GuessingGame [argument]");  
              System.out.println();  
              System.out.println ("Enter a number from 1 - 5 as your guess.");  
          }  
          else {  
              randomNum = ((int) (Math.random()\*5) + 1);  
              guess = Integer.parseInt(args[0]);
* To generate the random number 1 – 5, use this code.   
  *The relevant code is:  
    
  randomNum = ((int) (Math.random()\*5) + 1);*   
    
  **Code**  
  public static void main(String[] args) {  
       int randomNum = 0;  
          int guess;  
            
          if (args.length == 0 || args[0].compareTo("help") == 0){  
              System.out.println ("Usage: java GuessingGame [argument]");  
              System.out.println();  
              System.out.println ("Enter a number from 1 - 5 as your guess.");  
          }  
          else {  
              randomNum = ((int) (Math.random()\*5) + 1);  
              guess = Integer.parseInt(args[0]);
* Convert the runtime argument to an int before assigning it to the guess variable. Use the Integer.parseInt method to do the conversion.   
  *The relevant code is:  
    
  guess = Integer.parseInt(args[0]);*   
    
  **Code**  
  public static void main(String[] args) {  
       int randomNum = 0;  
          int guess;  
            
          if (args.length == 0 || args[0].compareTo("help") == 0){  
              System.out.println ("Usage: java GuessingGame [argument]");  
              System.out.println();  
              System.out.println ("Enter a number from 1 - 5 as your guess.");  
          }  
          else {  
              randomNum = ((int) (Math.random()\*5) + 1);  
              guess = Integer.parseInt(args[0]);

If the first argument in the args array equals "help" or if the args array is empty, display the usage of the program.

Graphic

*The relevant code snippets are:  
  
"Usage: java GuessingGame [argument]"  
  
and   
  
"Enter a number from 1 - 5 as your guess."*

Code

public static void main(String[] args) {  
     int randomNum = 0;  
        int guess;  
          
        if (args.length == 0 || args[0].compareTo("help") == 0){  
            System.out.println ("Usage: java GuessingGame [argument]");  
            System.out.println();  
            System.out.println ("Enter a number from 1 - 5 as your guess.");  
        }  
        else {  
            randomNum = ((int) (Math.random()\*5) + 1);  
            guess = Integer.parseInt(args[0]);

If a 1, 2, 3, 4, or 5 is entered, generate a random number. Then do the following:

Graphic

*This is the relevant code:  
  
randomNum = ((int) (Math.random()\*5) + 1);*

Code

public static void main(String[] args) {  
     int randomNum = 0;  
        int guess;  
          
        if (args.length == 0 || args[0].compareTo("help") == 0){  
            System.out.println ("Usage: java GuessingGame [argument]");  
            System.out.println();  
            System.out.println ("Enter a number from 1 - 5 as your guess.");  
        }  
        else {  
            randomNum = ((int) (Math.random()\*5) + 1);  
            guess = Integer.parseInt(args[0]);  
  
            if (guess < 1 || guess > 5) {  
                System.out.println ("Invalid argument: Enter a value between 1 and 5 only");  
            }  
...

* convert the arg[0] to an int and assign it to the guess variable, and   
  *This is the relevant code:  
    
  guess = Integer.parseInt(args[0]);*   
    
  **Code**  
  public static void main(String[] args) {  
       int randomNum = 0;  
          int guess;  
            
          if (args.length == 0 || args[0].compareTo("help") == 0){  
              System.out.println ("Usage: java GuessingGame [argument]");  
              System.out.println();  
              System.out.println ("Enter a number from 1 - 5 as your guess.");  
          }  
          else {  
              randomNum = ((int) (Math.random()\*5) + 1);  
              guess = Integer.parseInt(args[0]);  
    
              if (guess < 1 || guess > 5) {  
                  System.out.println ("Invalid argument: Enter a value between 1 and 5 only");  
              }  
  ...
* compare the guess to randomNum using a nested if/else construct   
  *This is the relevant code:  
    
  if (guess < 1 || guess > 5) {  
                 System.out.println ("Invalid argument: Enter a value between 1 and 5 only");  
             }  
             else {  
                 if (guess == randomNum) {*   
    
  **Code**  
              if (guess < 1 || guess > 5) {  
                  System.out.println ("Invalid argument: Enter a value between 1 and 5 only");  
              }  
              else {  
                  if (guess == randomNum) {  
                      System.out.println ("Great guess!  You got it right!");                  
                  }  
                  else {  
                      System.out.println("Sorry. The number was " + randomNum + ". Try again.");  
                  } // end of innermost nested if/else  
              } // end of first nested if/else  
          } // end of outer if/else                       
      } // end of main method

If they match, display a "congratulations" message.

Graphic

*The relevant code is:  
  
System.out.println ("Great guess!  You got it right!");*

Code

...  
            if (guess < 1 || guess > 5) {  
                System.out.println ("Invalid argument: Enter a value between 1 and 5 only");  
            }  
            else {  
                if (guess == randomNum) {  
                    System.out.println ("Great guess!  You got it right!");                  
                }  
                else {  
                    System.out.println("Sorry. The number was " + randomNum + ". Try again.");  
                } // end of innermost nested if/else  
            } // end of first nested if/else  
        } // end of outer if/else                       
    } // end of main method

If they don't, tell them what the random number was and ask them to "try again."

Graphic

*The relevant code is:  
  
System.out.println("Sorry. The number was "+ randomNum + ". Try again.");*

Code

...  
            if (guess < 1 || guess > 5) {  
                System.out.println ("Invalid argument: Enter a value between 1 and 5 only");  
            }  
            else {  
                if (guess == randomNum) {  
                    System.out.println ("Great guess!  You got it right!");                  
                }  
                else {  
                    System.out.println("Sorry. The number was " + randomNum + ". Try again.");  
                } // end of innermost nested if/else  
            } // end of first nested if/else  
        } // end of outer if/else                       
    } // end of main method

You can now save and compile the program and test it by running the GuessingGame class.  
  
Since no runtime parameter was passed to the args array, you should get this Usage message.

Graphic

*The output reads as follows:  
  
run:  
  
Usage: java GuessingGame [argument]  
  
Enter a number from 1 - 5 as your guess.  
  
BUILD SUCCESSFUL (total time: 0 seconds)*

Note

*When using an Integrated Development Environment, or IDE, you don't have access to the command line to provide runtime parameters. Therefore, you will enter your "guess" – or runtime parameter – as a runtime property of the project and then run the entire project, rather than just running an individual file.*

You can now right-click the project name in the Projects window and select **Properties** from the menu. Then you can follow these steps:

* select the **Run** category in the Project Properties window   
  *The Project Properties - Practice 08 dialog box contains a Categories pane, which contains the following categories: Sources, Libraries, Build, Run, Application, and Formatting.*
* change the Main Class to **GuessingGame**.   
  *The Browse Main Classes dialog box contains the Main classes list and the Select Main Class button. The Main classes list contains the following options: GuessingGame, NamesListTest, and VacationScaleTest.*
* enter a number between 1 and 5 in the Arguments field and click **OK**, and   
  *In this example, the number 2 is entered in the Arguments field.*
* run the project by clicking the **Run** button on the main toolbar

You should receive either the "congratulations" message or the "try again" message.  
  
Continue to click the **Run** button to see the different random numbers generated and how the program responds by comparing it with your guess.

Graphic

*Two output examples are shown. The first example reads:  
  
run:  
Sorry. The number was 1. Try again.  
BUILD SUCCESSFUL (total time: 0 seconds)  
  
The second example reads:  
  
run:  
Great guess! You got it right!  
BUILD SUCCESSFUL (total time: 0 seconds)*

**3. Summary**

In this topic, you've learned how to work with the ArrayList class.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#top)

Introducing Loop Constructs

Learning Objectives

*After completing this topic, you should be able to*

* *develop while and for loops*
* *work with while loops*

**1. Creating while and for loops**

In computer programming, it is a common task to need to repeat a number of statements. Typically, the code will continue to repeat the statements until something changes. Then, the code breaks out of the loop and continues with the next statement.  
  
Loops are often used in programs to repeat blocks of statements until an expression is false.

Code

while (!areWeThereYet) {  
  
    read book;  
    argue with sibling;  
    ask, "Are we there yet?";  
  
}  
  
Woohoo!;  
Get out of car;;

There are three main types of loops:

Code

while (!areWeThereYet) {  
  
    read book;  
    argue with sibling;  
    ask, "Are we there yet?";  
  
}  
  
Woohoo!;  
Get out of car;;

**while**

The while loop repeats while an expression is true.

**do/while, and**

The do/while loop executes once, and then continues to repeat while true.

**for**

The for loop repeats a set number of times.

This is the syntax for creating while loops.

Code

while (boolean\_expression) {  
  
    code\_block;  
  
} // end of while construct  
  
// program continues here

**code\_block;**

*If the Boolean expression is true, this code block will execute.*

**// program continues here**

*If the Boolean expression is false, the program will continue from the next block.*

The code shows a very simple while loop in the Elevator class. This elevator accepts commands only for going up or down one floor at a time. So in order to move a number of floors, the goUp()or goDown()methods need to be called a number of times.  
  
Notice how the Boolean expression is written. The expression returns true if currentFloor is not equal to desiredFloor. So, when these two variables are equal, this expression will return false because the elevator is now at the desired floor, and the while loop will not be executed.

Code

public void setFloor() {  
// Normally you would pass the desiredFloor as an argument to the  
// setFloor method. However, because you have not learned how to  
// do this yet, desiredFloor is set to a specific number (5)  
// below.  
  
   int desiredFloor = 5;  
   while ( currentFloor != desiredFloor ){  
      if (currentFloor < desiredFloor) {  
         goUp();  
      }  
      else {  
         goDown();  
      }  
   }  
}

This setFloor method uses two different types of variables. The variable currentFloor is an instance variable, usually called a field. It is a member of the class. Fields are declared outside of method code, usually just after the class declaration.

Code

public class Elevator {  
   public boolean doorOpen=false;  
   public int currentFloor = 1;  
   public final int TOP\_FLOOR = 5;  
   public final int BOTTOM\_FLOOR = 1;  
  
   ... < lines of code omitted > ...  
  
   public void setFloor() {  
      int desiredFloor = 5;  
      while ( currentFloor != desiredFloor ){  
        if (currentFloor < desiredFloor) {  
           goUp();  
        } else {  
           goDown();  
        }  
      } // end of while loop  
    } // end of method  
} // end of class

The variable desiredFloor is a local variable, declared within the setFloor method, and only  
accessible within that method. Another way to say this is that its scope is the setFloor method.  
  
Local variables can also be declared within loops or if statements. Irrespective of whether declared within a method, loop, or if statement, the scope of a local variable is always the block within which it is declared.

Code

public class Elevator {  
   public boolean doorOpen=false;  
   public int currentFloor = 1;  
   public final int TOP\_FLOOR = 5;  
   public final int BOTTOM\_FLOOR = 1;  
  
   ... < lines of code omitted > ...  
  
   public void setFloor() {  
     int desiredFloor = 5;  
     while ( currentFloor != desiredFloor ){  
       if (currentFloor < desiredFloor) {  
        goUp();  
       } else {  
        goDown();  
       }  
     } // end of while loop  
   } // end of method  
} // end of class

The example shows some code for generating the square root of a number. The Boolean expression squares the current value of the square root and checks to see if it is close to the number we are trying to find the square root of.  
  
If it's close enough and the expression returns true, the program execution skips the statements in the while block and continues with the System.out.println()statement that outputs the square root.

Code

float square = 4;    // number to find sq root of  
float squareRoot = square;    // first guess  
while (squareRoot \* squareRoot - square > 0.001) { // How accurate?  
    squareRoot = (squareRoot + square/squareRoot)/2;  
    System.out.println("Next try will be " + squareRoot);  
}  
System.out.println("Square root of " + square + " is " + squareRoot);

If the value is not yet close enough, the code within the block runs. It adjusts the value of squareRoot, so that it will be closer the next time it's checked. It also prints out the current "guessed" value of squareRoot.

Code

float square = 4;    // number to find sq root of  
float squareRoot = square;    // first guess  
while (squareRoot \* squareRoot - square > 0.001) { // How accurate?  
    squareRoot = (squareRoot + square/squareRoot)/2;  
    System.out.println("Next try will be " + squareRoot);  
}  
System.out.println("Square root of " + square + " is " + squareRoot);

Each result is shown until the correct answer is reached.

Code

Next try will be 2.5  
Next try will be 2.05  
Next try will be 2.0006099  
Next try will be 2.0  
The square root of 4.0 is 2.0

Here's an example that shows how long it would take to double your money at a particular interest  
rate. The while loop's Boolean expression checks to see if your money, which is converted to pennies, has doubled.  
  
If it hasn't, the block of the loop adds the interest of another year on to the current total and the loop repeats the Boolean expression check.

Code

int initialSum = 500;  
int interest = 7;        // per cent  
int years = 0;  
int currentSum = initialSum \* 100; // Convert to pennies  
   while ( currentSum <= 100000 ) {  
       currentSum += currentSum \* interest/100;  
       years++;  
       System.out.println("Year " + years + ": " + currentSum/100);  
    }

Note

*Converting to pennies is done to simplify the example so that the int type can be used.*

The results are displayed here. The while loop iterates 11 times before the Boolean test evaluates to true.

Code

... <some results not shown> ...  
Year 9: 919  
Year 10: 983  
Year 11: 1052

Loops are often used to repeat a set of commands a specific number of times. You can easily do this by declaring and initializing a counter, usually of type int, incrementing that variable inside the loop, and checking if the counter has reached a specific value in the while Boolean expression.  
  
However, although this works, Java has a special counter loop called a for loop.

Code

System.out.println(" /\*");  
int counter = 0;  
while ( counter < 4 ) {  
   System.out.println(" \*");  
   counter ++;  
}  
System.out.println(" \*/");

This is the output of the code.

Code

/\*  
\*  
\*  
\*  
\*  
\*/

A while loop differs in structure when compared to a for loop. The while loop opens with a counter variable on the first line. The boolean expression is on the second line, and the counter increment is on the fourth line of the loop.

Code

int counter = 0;  
while ( counter < 4 ) {  
   System.out.println(" \*");  
   counter ++;  
}

In the for loop, the three expressions needed for a loop that runs a set number of times are all moved into the parentheses after the for keyword. This makes the for loop more compact and readable.

Code

for ( int counter = 0 ; counter < 4 ; counter++ ) {  
  
   System.out.println(" \*");  
  
}

Notice that for loops are very versatile; you can initialize more than one variable in the first part,  
and modify more than one variable in the third part of the for statement. Also, the type need not be an int.  
  
The given code declares two Strings and, as it loops, appends to one String while removing from the other String. These changes are in the third part of the for statement. This part is for updates and, although often used for incrementing the String, can be used for any kind of update.

Graphic

*The third part of the for statement is:  
  
i += "|", t = t.substring(1) ) {*

Code

for (String i = "|", t = "------";  
     i.length() < 7 ;  
     i += "|", t = t.substring(1) ) {  
         System.out.println(i + t);  
}

Syntax

for (*initialize[,initialize];* *boolean\_expression;* *update[,update]*) {  
  
  
      *code\_block;*}

This is the output of the loop.

Code

|------  
||-----  
|||----  
||||---  
|||||--  
||||||-

You want to create the Counter class that uses a simple for loop to print a sequence of numbers.

You create a new Java class called "Counter." Declare and initialize a public final int field called MAX\_COUNT. Assign the value 100 to this field. You use the keyword final to designate this as a constant field.

Code

public final int MAX\_COUNT = 100;

You now need to create a method called displayCount that does the following:

* Counts from 1 to the value of the MAX\_COUNT constant, using a for loop. Increment the value of the loop variable by 1.
* Displays the value of the loop variable if it is divisible by 12. Display this on a single line, separated by a space.

Here is an example of a for loop.  
  
You can use the modulus operator, %, to check divisibility by 12. If it is divisible by 12, the result of the modulus operation will be zero. You can use the System.out.print method to keep all displayed values on the same line.

Code

for (int i= 1; i < 10; i++) // loops 9 times

The code for the displayCount method is displayed.

Code

public class Counter {  
  
    public final int MAX\_COUNT = 100;  
  
    public void displayCount() {  
        for (int count = 1; count <= MAX\_COUNT; count++) {  
            if (count % 12 == 0) {  
                System.out.print(count + " ");  
            } // end if  
        } // end for  
    }end method  
}

You save and compile your program and then test it by running the CounterTest class. You should receive the numbers 12 24 36 48 60 72 84 96 as an output.

Question

What is the output of this for loop?

**Code**  
for (String i = "|", t = "abcdefg";  
     i.length() < 7 ;  
     i += "|", t = t.substring(1) ) {  
         System.out.print(i +" "+ t);  
}

**Options:**

1. | abcdefg|| bcdefg||| cdefg|||| defg||||| efg||||||
2. fg efg|||||| defg||||| cdefg|||| bcdefg||| abcdefg|| |
3. |abcdefg||bcdefg|||cdefg||||defg|||||efg||||||fg
4. | abcdefg|| bcdefg||| cdefg|||| defg||||| efg|||||| fg

Answer

***Option 1:****Incorrect. The output is missing the fg from the final iteration of the loop.*

***Option 2:****Incorrect. The output here is the reverse of how it would be returned from the loop.*

***Option 3:****Incorrect. The print method includes a space between the results of each iteration of the loop, which is not included in this output.*

***Option 4:****Correct. This is the output that would be returned from this for loop, including a space between the results of each iteration of the loop.*

**Correct answer(s):**

4. | abcdefg|| bcdefg||| cdefg|||| defg||||| efg|||||| fg

**2. Working with while loops**

You now want to write a class that uses a while loop. This class is named Sequence and displays a sequence starting with the numbers 0 and 1. Successive numbers in the sequence are the sum of the previous two numbers, for example 0 1 1 2 3 5 8 13 21. This sequence is also called the Fibonacci series.

To do this, you first create a new Java class called Sequence with three fields called firstNumber, secondNumber, and nextNumber. Assign the values of 0 and 1 to the firstNumber and secondNumber fields, respectively. Also declare a public final int called SEQUENCE\_LIMIT and set its value to 100.

Code

public class Sequence {  
    public int firstNumber = 0;  
    public int secondNumber = 1;  
    public int nextNumber;  
    public final int SEQUENCE\_LIMIT = 100;

Then you create a method called displaySequence. You use the following steps to code the method:

Code

public void displaySequence(){  
    // Print the first two numbers  
    System.out.print(firstNumber + “ “);  
    System.out.print(secondNumber + “ “);  
    // Calculate the next number  
    nextNumber = firstNumber + secondNumber;  
  
    while(nextNumber <= SEQUENCE\_LIMIT){  
        // Print the next number of the sequence  
        System.out.print(nextNumber + “ “);  
        firstNumber = secondNumber; // new first number  
        secondNumber = nextNumber; // new second number  
        // Calculate the next potential number  
        nextNumber = firstNumber + secondNumber;  
    } // end of while  
    // Finish it off with a carriage return  
    System.out.println();  
} // end of method

* before the while loop begins, use the System.out.print method to print firstNumber and secondNumber, concatenating a space to the end of each variable in your print statements   
  *The relevant code is:  
    
     System.out.print(firstNumber + “ “);  
     System.out.print(secondNumber + “ “);*   
    
  **Code**  
  public void displaySequence(){  
      // Print the first two numbers  
      System.out.print(firstNumber + “ “);  
      System.out.print(secondNumber + “ “);  
      // Calculate the next number  
      nextNumber = firstNumber + secondNumber;  
    
      while(nextNumber <= SEQUENCE\_LIMIT){  
          // Print the next number of the sequence  
          System.out.print(nextNumber + “ “);  
          firstNumber = secondNumber; // new first number  
          secondNumber = nextNumber; // new second number  
          // Calculate the next potential number  
          nextNumber = firstNumber + secondNumber;  
      } // end of while  
      // Finish it off with a carriage return  
      System.out.println();  
  } // end of method
* set nextNumber equal to firstNumber + secondNumber, and   
  *The relevant code is:  
    
  nextNumber = firstNumber + secondNumber;*   
    
  **Code**  
  public void displaySequence(){  
      // Print the first two numbers  
      System.out.print(firstNumber + “ “);  
      System.out.print(secondNumber + “ “);  
      // Calculate the next number  
      nextNumber = firstNumber + secondNumber;  
    
      while(nextNumber <= SEQUENCE\_LIMIT){  
          // Print the next number of the sequence  
          System.out.print(nextNumber + “ “);  
          firstNumber = secondNumber; // new first number  
          secondNumber = nextNumber; // new second number  
          // Calculate the next potential number  
          nextNumber = firstNumber + secondNumber;  
      } // end of while  
      // Finish it off with a carriage return  
      System.out.println();  
  } // end of method
* start a while loop that evaluates this expression in determining whether to loop again   
  *The relevant code is:  
    
  while(nextNumber <= SEQUENCE\_LIMIT){*   
    
  **Code**  
  public void displaySequence(){  
      // Print the first two numbers  
      System.out.print(firstNumber + “ “);  
      System.out.print(secondNumber + “ “);  
      // Calculate the next number  
      nextNumber = firstNumber + secondNumber;  
    
      while(nextNumber <= SEQUENCE\_LIMIT){  
          // Print the next number of the sequence  
          System.out.print(nextNumber + “ “);  
          firstNumber = secondNumber; // new first number  
          secondNumber = nextNumber; // new second number  
          // Calculate the next potential number  
          nextNumber = firstNumber + secondNumber;  
      } // end of while  
      // Finish it off with a carriage return  
      System.out.println();  
  } // end of method

Within the while block, you

Code

public void displaySequence(){  
    // Print the first two numbers  
    System.out.print(firstNumber + “ “);  
    System.out.print(secondNumber + “ “);  
    // Calculate the next number  
    nextNumber = firstNumber + secondNumber;  
  
    while(nextNumber <= SEQUENCE\_LIMIT){  
        // Print the next number of the sequence  
        System.out.print(nextNumber + “ “);  
        firstNumber = secondNumber; // new first number  
        secondNumber = nextNumber; // new second number  
        // Calculate the next potential number  
        nextNumber = firstNumber + secondNumber;  
    } // end of while  
    // Finish it off with a carriage return  
    System.out.println();  
} // end of method

* print the nextNumber field and add a space to the end of it   
    
  **Code**  
  public void displaySequence(){  
      // Print the first two numbers  
      System.out.print(firstNumber + “ “);  
      System.out.print(secondNumber + “ “);  
      // Calculate the next number  
      nextNumber = firstNumber + secondNumber;  
    
      while(nextNumber <= SEQUENCE\_LIMIT){  
          // Print the next number of the sequence  
          System.out.print(nextNumber + “ “);  
          firstNumber = secondNumber; // new first number  
          secondNumber = nextNumber; // new second number  
          // Calculate the next potential number  
          nextNumber = firstNumber + secondNumber;  
      } // end of while  
      // Finish it off with a carriage return  
      System.out.println();  
  } // end of method
* set firstNumber equal to secondNumber, and secondNumber equal to nextNumber, and   
    
  **Code**  
  public void displaySequence(){  
      // Print the first two numbers  
      System.out.print(firstNumber + “ “);  
      System.out.print(secondNumber + “ “);  
      // Calculate the next number  
      nextNumber = firstNumber + secondNumber;  
    
      while(nextNumber <= SEQUENCE\_LIMIT){  
          // Print the next number of the sequence  
          System.out.print(nextNumber + “ “);  
          firstNumber = secondNumber; // new first number  
          secondNumber = nextNumber; // new second number  
          // Calculate the next potential number  
          nextNumber = firstNumber + secondNumber;  
      } // end of while  
      // Finish it off with a carriage return  
      System.out.println();  
  } // end of method
* set nextNumber equal to firstNumber + secondNumber   
    
  **Code**  
  public void displaySequence(){  
      // Print the first two numbers  
      System.out.print(firstNumber + “ “);  
      System.out.print(secondNumber + “ “);  
      // Calculate the next number  
      nextNumber = firstNumber + secondNumber;  
    
      while(nextNumber <= SEQUENCE\_LIMIT){  
          // Print the next number of the sequence  
          System.out.print(nextNumber + “ “);  
          firstNumber = secondNumber; // new first number  
          secondNumber = nextNumber; // new second number  
          // Calculate the next potential number  
          nextNumber = firstNumber + secondNumber;  
      } // end of while  
      // Finish it off with a carriage return  
      System.out.println();  
  } // end of method

Outside the while loop block, use System.out.println. This will create a new line for the "Build Successful" message that will appear after the sequence.

Graphic

*The relevant code is:  
  
// Finish it off with a carriage return  
System.out.println();*

Code

        while(nextNumber <= SEQUENCE\_LIMIT){  
            // Print the next number of the sequence  
            System.*out*.print(nextNumber + " ");  
            firstNumber = secondNumber; // new first number  
            secondNumber = nextNumber; // new second number  
            // Calculate the next potential number  
            nextNumber = firstNumber + secondNumber;  
        } // end of while  
        // Finish it off with a carriage return  
        System.*out*.println();  
    }   
}

Lastly, you save and compile your program and run the SequenceTest class to test it.

Graphic

*This is the output:  
  
run:  
0 1 1 2 3 5 8 13 21 34 55 89  
BUILD SUCCESSFUL (total time: 5 seconds)*

Question

You are writing an application that displays numbers only if they are less than 87.  
  
Which code segment completes the code to do this?

**Code**  
int num = 0;  
INSERT THE MISSING CODE     {  
  System.out.println("Num = " + num);  
  num++;  
}

**Options:**

1. while (num < 87)
2. while num++
3. while (num > 87)
4. while num < 87

Answer

***Option 1:****Correct. This segment completes the code so it will display numbers only if they are less than 87. Each number will appear on a separate line, prefaced with "Num = ".*

***Option 2:****Incorrect. The proper syntax for using a while loop is the while keyword followed by a Boolean expression enclosed in parentheses.*

***Option 3:****Incorrect. The Boolean expression in this code segment is incorrect. It should test for numbers that are less than 87 instead of numbers that are greater than 87.*

***Option 4:****Incorrect. The Boolean expression in this code segment should be enclosed in parentheses following the while keyword.*

**Correct answer(s):**

1. while (num < 87)

**3. Converting a while loop**

Here you wish to create a new class, ChallengeSequence, based upon the Sequence class. You need to modify the displaySequence method to use a for loop instead of a while loop.

Code

public class ChallengeSequence{  
    public int firstNumber = 0;  
    public int secondNumber = 1;  
    public int nextNumber;  
    public final int SEQUENCE\_LIMIT = 100;  
    public void displaySequence(){  
        // Print the first two numbers  
        System.out.print(firstNumber + “ “);  
        System.out.print(secondNumber + “ “);  
        // Calculate the next number  
        nextNumber = firstNumber + secondNumber;  
  
        while(nextNumber <= SEQUENCE\_LIMIT){  
            // Print the next number of the sequence  
            System.out.print(nextNumber + “ “);  
            firstNumber = secondNumber; // new first number  
            secondNumber = nextNumber; // new second number  
            // Calculate the next potential number  
            nextNumber = firstNumber + secondNumber;  
        } // end of while  
        // Finish it off with a carriage return  
        System.out.println();  
    } // end of method  
} // end of class

To begin, create a new Java class called ChallengeSequence. Copy all the code that occurs between the outer class brackets of the Sequence class.  
  
You then paste it inside the outer brackets of the ChallengeSequence class.

Code

public class ChallengeSequence{  
    public int firstNumber = 0;  
    public int secondNumber = 1;  
    public int nextNumber;  
    public final int SEQUENCE\_LIMIT = 100;   
    public void displaySequence(){  
        // Print the first two numbers  
        System.out.print(firstNumber + " ");  
        System.out.print(secondNumber + " ");  
        // Calculate the next number  
        nextNumber = firstNumber + secondNumber;  
        while(nextNumber <= SEQUENCE\_LIMIT){  
            System.out.print(nextNumber + " "); // Print the next number of the sequence  
            firstNumber = secondNumber; // new first number  
            secondNumber = nextNumber; // new second number  
           nextNumber = firstNumber + secondNumber; // Calculate the next potential number  
        } // end of while  
        System.out.println();// Finish it off with a carriage return  
    }   
}

Then create an additional final field called SEQUENCE\_COUNT and assign a value of 10 to it. Be sure that you don't change any of the other field names.

Code

public final int SEQUENCE\_COUNT = 10;

In the displaySequence method, modify the while loop to a for loop such that only the first 10 values of the Fibonacci series are displayed.  
  
Remember that the first two numbers in the sequence are displayed before the loop begins. Your for loop must display the remaining 8 values.  
  
There are a several ways of handling the discrepancy between the SEQUENCE\_COUNT value and the number of values that need to be displayed within the loop. One approach is to adjust the initial count in the loop. This is the solution used here.

Code

public void displaySequence(){  
        // Display the first two numbers of the sequence  
        System.out.print(firstNumber + " ");  
        System.out.print(secondNumber + " ");  
        // Calculate the next number of the sequence  
        nextNumber = firstNumber + secondNumber;  
          
        for (int count = 2; count < SEQUENCE\_COUNT; count++){  
            // Print the next number of the sequence  
            System.out.print(nextNumber + " ");  
            firstNumber = secondNumber; // new firstNumber  
            secondNumber = nextNumber; // new secondNumber  
            // calculate next potential number in sequence  
            nextNumber = firstNumber + secondNumber;   
        }  
        // Finish it off with a carriage return  
        System.out.println();    
    }

Supplement

*Selecting the link title opens the resource in a new browser window.*

**Job aid**

Access the job aid [ChallengeSequence class](javascript:doWindow('./ja_jsef_a03_it_enust1401_frame.html')) to review the source code for the ChallengeSequence class used here.

You save and compile your program. Then run the ChallengeSequenceTest class to test your code. This is the output.

Code

0 1 1 2 3 5 8 13 21 34

This code shows a simple nested loop to output a block of @ symbols with height and width given in the initial local variables. Notice how the outer code prints a new line to start a new row, while the inner loop uses the print() method of System.out to print an @ symbol for every column.

Code

int height = 4;  
int width = 10;  
  
for (int rowCount = 0; rowCount < height; rowCount++ ) {  
  
    for (int colCount = 0; colCount < width; colCount++ ) {  
        System.out.print("@");  
    }  
    System.out.println();  
}

Here's a nested while loop that's a little more complex. The nested loop tries to guess a name by building a String of the same length completely at random.  
  
Looking at the inner loop first, the code initializes the char asciiChar to a lowercase letter randomly. These chars are then added to the String guess, until that String is as long as the String that it’s being matched against. Notice the convenience of the concatenation operator here, allowing concatenation of a String and a char.  
  
The outer loop tests to see if the guess is the same as a lowercase version of the original name. If it isn't, guess is reset to an empty String and the inner loop runs again, usually millions of times for a five-letter name.

Code

String name = "Lenny";  
String guess = "";  
int numTries = 0;  
  
while (!guess.equals(name.toLowerCase())) {  
    guess = "";  
    while (guess.length() < name.length()) {  
        char asciiChar = (char)(Math.random() \* 26 + 97);  
        guess = guess + asciiChar;  
    }  
    numTries++;  
}  
System.out.println(name + " found after " + numTries + " tries!");

Note

*Names that are longer than five letters will take a very long time.*

**4. Summary**

In this topic, you've learnt how to develop a for loop, and how to work with and convert while loops.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#top)

Using Loops

Learning Objective

*After completing this topic, you should be able to*

* *use an array in a loop and code and test a do/while loop*

**1. Using loops**

One of the most common uses of loops is when working with sets of data.  
  
All types of loops are useful:

* while loops, if checking for a particular value
* for loops, to go through the entire array, and
* enhanced for loops

A for loop can be used with an array. The Index starts at 0. In this example, the last index of the array is ages.length -1. The ages[i] counter accesses array values as i goes from 0 to ages.length -1.

Code

for (int i = 0; i < ages.length; i++ ) {  
   System.out.println("Age is " + ages[i] );  
}

The for loop returns this output.

Code

Age is 27  
Age is 12  
Age is 82  
…  
Age is 1

When you set values in an array, the loop accesses each element of the array in turn. In this example, each element in the array is set to 10.

Graphic

*The relevant code is:  
  
ages[i] = 10;*

Code

for (int i = 0; int < ages.length; i++ ) {  
   ages[i] = 10;  
}

When using an enhanced for loop with an array, the loop accesses each element of the array in turn. In this example, each iteration returns the next element of the array in age.

Code

for (int age : ages ) {  
   System.out.println("Age is " + age );  
}

When you use an enhanced for loop with ArrayLists, the loop accesses each element of ArrayList in turn. In this example, each iteration returns the next element of the ArrayList in name. ArrayLists can be iterated through in exactly the same way as arrays.

Code

for (String name : names ) {  
   System.out.println("Name is " + name);  
}

There are two useful keywords that can be used when working with loops.

**break**

The break keyword allows you to jump out of a loop. This example shows the use of break. Assuming that the code is to find out if any of the scores in the array are above the passmark, we can set passed to true and jump out of the loop as soon as the first such score is found.   
  
**Code**  
int passmark = 12;  
boolean passed = false;  
int[] score = { 4, 6, 2, 8, 12, 34, 9 };  
for (int unitScore : score ) {  
    if ( unitScore > passmark ) {  
       passed = true;  
       break;  
    }  
}  
System.out.println("One or more units passed? " + passed);

**continue**

The continue keyword sends you back to the start of the loop. This example shows the use of continue. In this case, we want to know if a certain number of passes has been achieved. The approach is to first check to see if the unit's score is not enough. If this is the case, the continue command goes to the start of the loop again. If the score is sufficient, the number of passesReqd is decremented and possibly further processing takes place.   
  
**Code**  
int passMark = 15;  
int passesReqd = 3;  
int[] score = { 4, 6, 2, 8, 12, 34, 9 };  
for (int unitScore : score ) {  
    if (unitScore < passMark) {  
        continue;  
    }  
    passesReqd--;  
    // Other processing  
}  
System.out.println("Units still reqd " + Math.max(0,passesReqd));

These examples are intended only to show what the functions of break and continue are, and not to show particular programming techniques. Both have a similar function. They ensure that parts of the loop are not processed unnecessarily. Sometimes this can also be achieved by the design of if blocks, but in complex algorithms it is useful to have these two options.

The do/while loop is a one-to-many iterative loop: The condition is at the bottom of the loop and is processed after the body. The body of the loop is therefore processed at least once. If you want the statement, or statements, in the body to be processed at least once, use a do/while loop instead of a while or for loop.

Syntax

do {  
  
        *code\_block;*}  
while (*boolean\_expression*); // Semicolon is mandatory.

The setFloor method of the Elevator class uses a do/while loop to determine if the elevator  
is at the chosen floor. If the value of the currentFloor variable is not equal to the value of the desiredFloor variable, the elevator continues moving either up or down.

Code

setFloor() {  
     // Normally you would pass the desiredFloor as an argument to the  
    // setFloor method. However, because you have not learned how to  
    // do this yet, desiredFloor is set to a specific number (5)  
    // below.  
      int desiredFloor = 5;  
  
  
    do {  
      if (currentFloor < desiredFloor) {  
        goUp();  
      }  
      else if (currentFloor > desiredFloor) {  
        goDown();  
      }  
     }  
      while (currentFloor != desiredFloor);  
  }

The different loop constructs each have different uses:

**The while loop**

The while loop is used to iterate indefinitely through statements and to perform the statements zero or more times.

**The do/while loop, and**

The do/while loop is used to iterate indefinitely through statements and to perform the statements one or more times.

**The for loop**

The for loop is used to step through statements a predefined number of times.

Question

What type of loop is a one-to-many iterative loop?

**Options:**

1. The while loop
2. The nested while loop
3. The do/while loop
4. The for loop

Answer

***Option 1:****Incorrect. A while loop repeats while an expression is true. You use the while loop to iterate indefinitely through statements and to perform the statements zero or more times.*

***Option 2:****Incorrect. When the body of a loop contains another loop, it is known as a nested loop. For example, a nested loop can be used to print a table with rows and columns.*

***Option 3:****Correct. The do/while loop is a one-to-many iterative loop. The condition is at the bottom of the loop and is processed after the body. The body of the loop is therefore processed at least once.*

***Option 4:****Incorrect. A for loop repeats a set number of times. With a for loop, you can initialize more than one variable in the first part, and modify more than one variable in the third part of the for statement.*

**Correct answer(s):**

3. The do/while loop

You'll now create two new methods in two different classes. One of the methods uses a traditional for loop to display the values in an ArrayList. The other method uses an enhanced for loop to display the values in the ArrayList.

You want to use a for loop with the VacationScaleTwo class. You use the VacationScaleTwo class which has already been partially coded. This is similar to the VacationScale class, but an ArrayList is used to store vacation days instead of an array.

Code

public class VacationScaleTwo {  
    public ArrayList vacationDays;  
      
    public void setVacationScale(){  
        vacationDays = new ArrayList(7);  
        vacationDays.add(10);  
        vacationDays.add(15);  
        vacationDays.add(15);  
        vacationDays.add(15);  
        vacationDays.add(20);  
        vacationDays.add(20);  
        vacationDays.add(25);  
    }  
}

You need to add a new method called displayVacationDays. In the displayVacationDays method, you add a for loop.

Code

    public void displayVacationDays(){  
        for (int years = 0; years < vacationDays.size(); years++){  
              
  
        }// end for loop

In the for loop block, use System.out.println to print the value of each ArrayList element. Then use the get method of the ArrayList, passing the years variable as an argument. It references the current index number of the vacationDays list.

Code

    public void displayVacationDays(){  
        for (int years = 0; years < vacationDays.size(); years++){  
            System.out.println("The vacation for " + years +   
                               " years of service is: "   
                     + vacationDays.get(years) );  
        }// end for loop  
    } // end method

You save and compile your program, and then code the VacationScaleTwoTest class.

Code

public class VacationScaletwoTest {  
     
  public static void main (String args[]) {  
   
  VacationScale myVacationScale = new VacationScale();  
    
  myVacationScale.setVacationScale();  
  myVacationScale.yearsOfService = 1;  
  myVacationScale.displayVacationDays();  
    
  myVacationScale.yearsOfService = 5;  
  myVacationScale.displayVacationDays();  
    
  myVacationScale.yearsOfService = 10;  
  myVacationScale.displayVacationDays();  
 }   
}

You then run the VacationScaleTwoTest class to test your program.

Graphic

*This is the output.  
  
run:   
The vacation for 0 years of service is: 10  
The vacation for 1 years of service is: 15  
The vacation for 2 years of service is: 15  
The vacation for 3 years of service is: 15  
The vacation for 4 years of service is: 20  
The vacation for 5 years of service is: 20  
The vacation for 6 years of service is: 25  
BUILD SUCCESSFUL (total time: 0 seconds)*

You want to use an enhanced for loop with the NamesListTwo class.  
  
You move on to open the NamesListTwo class which has already been partially coded. It has only one method, setList, that initializes the ArrayList and then prints the size of the list.

Code

public class NamesListTwo {  
    public ArrayList names;  
      
    public void setList(){  
        names = new ArrayList();  
        names.add("Joe Smith");  
        names.add("Mary Palmer");  
        names.add("Jose Gonzalez");  
        names.add("Linda Baker");  
        System.out.println("Size of ArrayList is: " + names.size());  
    }  
         
}

You add a new method to the NamesListTwo class called displayNames. You will use an enhanced for loop in this method to process the ArrayList. To code the method, you do the following:

Code

public void displayNames(){

* use the System.out.println method to print this message   
  *This is the relevant code:  
    
  System.out.println("Names in the ArrayList: ");*   
    
  **Code**  
  public void displayNames(){  
          System.out.println("Names in the ArrayList: ");
* start the enhanced for loop, and   
  *This is the relevant code:  
    
  (Object name : names) is highlighted*   
    
  **Code**  
  public void displayNames(){  
          System.out.println("Names in the ArrayList: ");  
          for (Object name : names)
* within the for loop block, use  
  System.out.println to print the name reference   
  *This is the relevant code:  
    
  System.out.println(name); is highlighted*   
    
  **Code**  
  public void displayNames(){  
          System.out.println("Names in the ArrayList: ");  
          for (Object name : names)  
              System.out.println(name);  
          }  
      }

You then create a new Java Main Class called NamesListTwoTest.  
  
In the main method, you do the following:

Code

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

* declare and initialize a local variable of type NamesListTwo called namesList   
  *The relevant code is:  
    
  NamesListTwo namesList = new NamesListTwo();*   
    
  **Code**  
  public class NamesListTwoTest {    
      public static void main(String[] args) {  
          NamesListTwo namesList = new NamesListTwo();
* invoke the setList method of the namesList object, and   
  *The relevant code is:  
    
  namesList.setList();*   
    
  **Code**  
  public class NamesListTwoTest {  
      public static void main(String[] args) {  
          NamesListTwo namesList = new NamesListTwo();  
          namesList.setList();
* invoke the displayNames method of the namesList object   
  *The relevant code is:  
    
  namesList.displayNames();*   
    
  **Code**  
  public class NamesListTwoTest {   
      public static void main(String[] args) {  
          NamesListTwo namesList = new NamesListTwo();  
          namesList.setList();  
          namesList.displayNames();  
        
     }   
  }

Save and compile your program. Run the NamesListTwoTest class to test it.  
  
This is the output from the program.

Graphic

*This is the output:  
  
run:   
Size of ArrayList is: 4  
Names in the ArrayList:  
Joe Smith  
Mary Palmer  
Jose Gonzalez  
Linda Baker  
BUILD SUCCESSFUL (total time: 0 seconds)*

**2. Using a nested for loop**

You'll now create and process a two-dimensional array using a nested for loop. A nested loop is one loop within another loop.   
  
This example is based on the scenario of a classroom. A classroom has 12 desks arranged in a rectangular grid comprised of three rows and four columns. Students are allocated a desk at the position found vacant first, by traversing each row. The table shows the class map as a grid. Each cell represents a desk. Each cell contains the coordinates of the desk position in the class map.

Graphic

*The table is made up of four columns and three rows. Column 1, row 1 reads 0,0, row 2 reads 1,0, and row 3 reads 2,0. Column 2, row 1 reads 0,1, row 2 reads 1,1, and row 3 reads 2,1. Column 3, row 1 reads 0,2, row 2 reads 1,2, and row 3 reads 2,2. Column 4, row 1 reads 0,3, row 2 reads 1,3, and row 3 reads 2,3.*

You need to create a new Java class called ClassMap. In the class, you declare two public fields.

Code

public String[][] deskArray;  
public String name;

Then create a new method called setClassMap. In this method, you need to initialize the deskArray to have three rows and four columns.

Code

deskArray = new String[3][4];

Now create another new method called setDesk. This method will assign a new student, identified by the name field, which will be set by the ClassMapTest, to an empty desk in the class map.

Define the method using the following steps:

* traverse the deskArray to identify the first vacant element in it using a nested for loop   
    
  **Code**  
  public void setDesk() {  
      boolean flag = false;  
      for (int row = 0; row < 3; row++) { // start of row loop  
          for (int col = 0; col < 4; col++) { // start of column loop  
              if (deskArray[row][col] == null) {
* assign the value of the name field to the vacant element, if you find a null value in the deskMap, and   
    
  **Code**  
  public void setDesk() {  
      boolean flag = false;  
      for (int row = 0; row < 3; row++) { // start of row loop  
          for (int col = 0; col < 4; col++) { // start of column loop  
              if (deskArray[row][col] == null) {  
                  deskArray[row][col] = name;
* print the position of the desk for the student and exit out of the loops using a break statement to branch out of a running loop   
    
  **Code**  
  public void setDesk() {  
      boolean flag = false;  
      for (int row = 0; row < 3; row++) { // start of row loop  
          for (int col = 0; col < 4; col++) { // start of column loop  
              if (deskArray[row][col] == null) {  
                  deskArray[row][col] = name;  
                  System.out.println(name +  
      " desk set at position: Row:" +  
                            row + " Column:" + col);  
                  flag = true;  
                  break;  
              } // end of if  
          } // end of inner/column loop  
          if (flag == true) {  
              break;  
          } // end of if  
      } // end of outer/row loop  
      if (flag == false) {  
          System.out.println("All desks occupied.");  
      } // end of if  
  } //end method

Move on to create another new method called displayDeskMap. In this method, traverse the deskArray in the same way you did in the last step. For each element in the array, print the name in that element, or print "null." The output should be in the form of grid.  
  
You save and compile your program.

Graphic

*This is an example of what the output might look like. There are four names in the top elements in the array. These names are Ann, Bond, Cindy, and Donald. Below each name are two null entries.*

Code

public void displayDeskMap() {  
    for (int row = 0; row < 3; row++) {  
        for (int col = 0; col < 4; col++) {  
            System.out.print("  " + deskArray[row][col] + "  ");  
        }  
        System.out.println();  
    }  
}//end method

Then open The main method the ClassMapTest class first calls setClassMap to initialize the array. Next it assigns a value to the name field of the myClassMap object and then invokes setDesk. It does this four times, with a different name value each time.

Code

public static void main(String args[]){  
    // creating ChallengeClassMap instance  
    ClassMap cm = new ClassMap();  
      
    // Initialize the array  
    cm.setClassMap();  
  
    // Allocating position to one student  
    cm.name = "Ann";  
    cm.setDesk();  
  
    //Allocating position to another student  
    cm.name = "Bond";  
    cm.setDesk();  
  
    //Allocating position to third student  
    cm.name = "Cindy";  
    cm.setDesk();  
  
    //Allocating position to fourth student  
    cm.name = "Donald";  
   cm.setDesk();

Finally, it invokes displayDeskMap.

Code

     //Displaying the map of the class   
    cm.displayDeskMap();  
    

Run the ClassMapTest class to test your program.

Graphic

*This is the output:  
  
run:  
Ann  desk set at position: Row:0 Column:0  
Bond  desk set at position: Row:0 Column:1  
Cindy  desk set at position: Row:0 Column:2  
Donald  desk set at position: Row:0 Column:3  
 Ann    Bond    Cindy    Donald  
 null   null    null     null  
 null   null    null     null  
BUILD SUCCESSFUL (total time: 0 seconds)*

Question

Suppose you want to iterate through the multi-dimensional array eventsAndVenues and store the results of the top-level array in results.  
  
Which section of code iterates through the top level of the array?

**Options:**

1. for (Int[] results : eventsAndVenues)
2. for (Object[] eventsAndVenues : results)
3. for (Object results : eventsAndVenues)
4. for (Object[] results : eventsAndVenues)

Answer

***Option 1:****Incorrect. You always specify an Object type for multi-dimensional arrays because the elements in the top-level array are objects.*

***Option 2:****Incorrect. The iteration variable and array are reversed. You first specify the iteration variable and then the array.*

***Option 3:****Incorrect. This line of code iterates through the second-level of the array.*

***Option 4:****Correct. You set the type to Object because the top-level array contains objects, specify the iteration variable to results, and then specify to the eventsAndVenues array.*

**Correct answer(s):**

4. for (Object[] results : eventsAndVenues)

**3. Adding a Search Method**

You need to add another method to the ClassMap class. This method searches through the deskArray to find a certain name.   
  
You begin by adding a new method called searchDesk in the ClassMap class.

Code

public void searchDesk() {

In the searchDesk method, you do the following:

Code

public void searchDesk() {  
    boolean flag= false;  
    for(int row=0; row<3; row++){  
        for(int col=0; col<4; col++){  
            if(deskArray[row][col] != null &&  
                deskArray[row][col].equals(name)){  
                    System.out.println  
                   (name +” Desk Position: Row: ”+row+” Column: “  
                       +col);  
                flag = true;  
                break;  
            } // end of if  
        } // end of column loop  
        if (flag == true){  
            break;  
        } // end of if  
    } // end of row loop  
    if (flag == false){  
        System.out.println(“Desk not allocated for ”+name);  
    } // end of if  
} // end of method

* create a nested for loop to traverse through the deskArray   
  *The relevant code is:  
    
       
  for(int row=0; row<3; row++){  
  for(int col=0; col<4; col++){*   
    
  **Code**  
  public void searchDesk() {  
      boolean flag= false;  
      for(int row=0; row<3; row++){  
          for(int col=0; col<4; col++){  
              if(deskArray[row][col] != null &&  
                  deskArray[row][col].equals(name)){  
                      System.out.println  
                     (name +” Desk Position: Row: ”+row+” Column: “  
                         +col);  
                  flag = true;  
                  break;  
              } // end of if  
          } // end of column loop  
          if (flag == true){  
              break;  
          } // end of if  
      } // end of row loop  
      if (flag == false){  
          System.out.println(“Desk not allocated for ”+name);  
      } // end of if  
  } // end of method
* compare the value of the name field with the value of the element, if the array element is not null   
  *The relevant code is:  
    
    
  if(deskArray[row][col] != null &&  
  deskArray[row][col].equals(name)){*   
    
  **Code**  
  public void searchDesk() {  
      boolean flag= false;  
      for(int row=0; row<3; row++){  
          for(int col=0; col<4; col++){  
              if(deskArray[row][col] != null &&  
                  deskArray[row][col].equals(name)){  
                      System.out.println  
                     (name +” Desk Position: Row: ”+row+” Column: “  
                         +col);  
                  flag = true;  
                  break;  
              } // end of if  
          } // end of column loop  
          if (flag == true){  
              break;  
          } // end of if  
      } // end of row loop  
      if (flag == false){  
          System.out.println(“Desk not allocated for ”+name);  
      } // end of if  
  } // end of method
* print the position of the desk if the names are equal   
  *The relevant code is:  
    
    
  System.out.println  
  (name +” Desk Position: Row: ”+row+” Column: “ +col);*   
    
  **Code**  
  public void searchDesk() {  
      boolean flag= false;  
      for(int row=0; row<3; row++){  
          for(int col=0; col<4; col++){  
              if(deskArray[row][col] != null &&  
                  deskArray[row][col].equals(name)){  
                      System.out.println  
                     (name +” Desk Position: Row: ”+row+” Column: “  
                         +col);  
                  flag = true;  
                  break;  
              } // end of if  
          } // end of column loop  
          if (flag == true){  
              break;  
          } // end of if  
      } // end of row loop  
      if (flag == false){  
          System.out.println(“Desk not allocated for ”+name);  
      } // end of if  
  } // end of method
* print an error message if the name is not found in the deskArray, and   
  *The relevant code is:  
    
  if (flag == false) {  
  System.out.println(“Desk not allocated for ”+name);*   
    
  **Code**  
  public void searchDesk() {  
      boolean flag= false;  
      for(int row=0; row<3; row++){  
          for(int col=0; col<4; col++){  
              if(deskArray[row][col] != null &&  
                  deskArray[row][col].equals(name)){  
                      System.out.println  
                     (name +” Desk Position: Row: ”+row+” Column: “  
                         +col);  
                  flag = true;  
                  break;  
              } // end of if  
          } // end of column loop  
          if (flag == true){  
              break;  
          } // end of if  
      } // end of row loop  
      if (flag == false){  
          System.out.println(“Desk not allocated for ”+name);  
      } // end of if  
  } // end of method
* use the break statement to branch or exit out of the loops wherever required   
  *The relevant code is:  
    
  break;*   
    
  **Code**  
  public void searchDesk() {  
      boolean flag= false;  
      for(int row=0; row<3; row++){  
          for(int col=0; col<4; col++){  
              if(deskArray[row][col] != null &&  
                  deskArray[row][col].equals(name)){  
                      System.out.println  
                     (name +” Desk Position: Row: ”+row+” Column: “  
                         +col);  
                  flag = true;  
                  break;  
              } // end of if  
          } // end of column loop  
          if (flag == true){  
              break;  
          } // end of if  
      } // end of row loop  
      if (flag == false){  
          System.out.println(“Desk not allocated for ”+name);  
      } // end of if  
  } // end of method

In the ClassMapTest class, add these lines of code that set the name value of myClassMap object and invoke its searchDesk method. This combination occurs twice.

Code

    //cm.name = "Donald";  
    //cm.searchDesk();  
  
    //cm.name = "Ronn";  
    //cm.searchDesk();

You then save and compile your program. Run the ClassMapTest class to test the program.

Graphic

*Practice 09 is open in NetBeans IDE 7.0.1. ClassMapTest.java is open in the editor panel and has been run. The output is displayed in the Output panel:   
  
run:  
Ann  desk set at position: Row:0 Column:0  
Bond  desk set at position: Row:0 Column:1  
Cindy  desk set at position: Row:0 Column:2  
Donald  desk set at position: Row:0 Column:3  
 Ann    Bond    Cindy    Donald  
 null   null    null     null  
 null   null    null     null  
Donald  Desk Position: Row:   0 Column:   3  
Desk not allocated for: Ronn  
BUILD SUCCESSFUL (total time: 0 seconds)*

**4. Summary**

In this topic, you've learnt how to use an array in a loop, and code and test a do/while loop.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#top)

Using Loop Constructs and Arrays

Learning Objectives

*After completing this topic, you should be able to*

* *create and populate an array list*
* *manipulate an ArrayList*
* *use loops to process an ArrayList*

**1. Exercise overview**

In this exercise, you're required to use loops to process data within ArrayLists.

This involves the following tasks:

* creating and populating an ArrayList
* manipulating an ArrayList, and
* using loops to process an ArrayList

**2. Create and populate an ArrayList**

You want to update the company database to include a more detailed breakdown of the sick time logged by employees.

Question

You are writing a class to track the sick time of employees at your company. You decide to use an ArrayList to store the related data. However, after you declare the ArrayList variable in your class, you receive an error.  
  
Which statements would correct this error?

**Code**  
cannot find symbol   symbol:   class ArrayList   location: class SickDays

**Options:**

1. import java.util.+;
2. import java.util.ArrayList;
3. public java.util.Arraylist sickTime;
4. import java.util.\*;

Answer

***Option 1:****Incorrect. The use of the ".+" is not a valid way to import all the classes in the util package.*

***Option 2:****Correct. By adding the import statement to the top of the class and referring to the ArrayList class, you can now use this class in your program.*

***Option 3:****Incorrect. You can use a fully qualified reference to the class when declaring your variable. However, in this case, ArrayList is spelled incorrectly.*

***Option 4:****Correct. You can also add an import statement to the top of the class to refer to all the classes in the java.util package.*

**Correct answer(s):**

2. import java.util.ArrayList;  
4. import java.util.\*;

Question

Once you have declared the ArrayList object named sickTime correctly, you need to populate it with the sick time data.  
  
Which code segment completes the method and instantiates the ArrayList correctly?

**Code**  
public void setSickDays() {  
       INSERT THE MISSING CODE  
        sickTime.add(1);  
        sickTime.add(5);  
        sickTime.add(8);  
        sickTime.add(15);  
        sickTime.add(2);  
  
    }

**Options:**

1. sickTime = new ArrayList();
2. sickTime = new int [5];
3. sickTime = add ArrayList(0);
4. sickTime = new Array List(5);

Answer

***Option 1:****Correct. This syntax correctly instantiates an ArrayList object to store the sick time data.*

***Option 2:****Incorrect. This syntax would instantiate an array of type int, not an ArrayList.*

***Option 3:****Incorrect. This option uses the add method. This is used to insert data and add a new element to the end of the ArrayList.*

***Option 4:****Incorrect. While you can specify the initial length, this option does not correctly refer to ArrayList.*

**Correct answer(s):**

1. sickTime = new ArrayList();

**3. Manipulate an ArrayList**

You have created an ArrayList and need to display its contents.

Question

You now want to remove one of the entries from the ArrayList.  
  
Given that the list contains five elements, which code segments will remove the last item from the list?

**Options:**

1. sickTime.delete(5);
2. sickTime.remove(4);
3. sickTime.remove(sickTime.size());
4. sickTime.remove(sickTime.size()-1);

Answer

***Option 1:****Incorrect. Delete is not a valid method of ArrayList.*

***Option 2:****Correct. The remove method removes an item based on its element number. ArrayList elements begin at zero, so four would refer to the index of the last element in this case.*

***Option 3:****Incorrect. You need to subtract one from the number of elements to refer to the index of the proper element.*

***Option 4:****Correct. You can use size() to give you the number of elements of the array so the index of the last one is the size minus one.*

**Correct answer(s):**

2. sickTime.remove(4);  
4. sickTime.remove(sickTime.size()-1);

Question

You want to display the contents of the ArrayList using a method.  
  
Which code segment correctly completes the method and displays the results?

**Code**  
public void displaySickDays(){  
         INSERT THE MISSING CODE  
    } // end method

**Options:**

1. System.out.println(sickTime);
2. System.out.println(sickTime)
3. System.out.printArrayList(sickTime);
4. System.out.println(sickTime.Contents);

Answer

***Option 1:****Correct. You can use a reference to your ArrayList object to retrieve its contents.*

***Option 2:****Incorrect. While this option does use the ArrayList reference correctly, the missing semi-colon is a syntax error.*

***Option 3:****Incorrect. There is no printArrayList method defined.*

***Option 4:****Incorrect. Contents is not a valid property of the ArrayList object.*

**Correct answer(s):**

1. System.out.println(sickTime);

**4. Use loops to process an ArrayList**

You need to complete the new sick leave database application using loops to process the data in the ArrayList.

Question

You decide to use a for loop to output the contents of the ArrayList sickTime.  
  
Which code segment correctly completes the loop so you can display the contents?

**Code**  
for (INSERT THE MISSING CODE){  
          System.out.println("The sick time for employee" + emp +  
              " is: " + sickTime.get(emp) );  
        }// end for loop

**Options:**

1. int emp = 0; emp < sickTime.size();emp++
2. int emp = 0; emp < sickTime.size()-1;emp++
3. int emp = 0; emp > sickTime.size();emp++
4. int emp = 0; emp < sickTime.size();emp+

Answer

***Option 1:****Correct. This segment contains the three expressions needed for a loop that runs a set number of times.*

***Option 2:****Incorrect. This segment does not correctly calculate the number of iterations you'll need for the Boolean expression. This segment will stop one short of the required number of iterations.*

***Option 3:****Incorrect. This Boolean expression contains a logic error. Since emp is initially set to zero, this expression will never evaluate to true.*

***Option 4:****Incorrect. While this segment contains all the required elements, it does not increment the counter correctly.*

**Correct answer(s):**

1. int emp = 0; emp < sickTime.size();emp++

Question

To save time and space, you decide to utilize an enhanced for loop to iterate through and print the contents of the ArrayList.  
  
Which code segment correctly completes the enhanced for loop?

**Code**  
INSERT THE MISSING CODE {  
            System.out.println(sTime);  
         }

**Options:**

1. for (Object sTime : sickTime)
2. For (Object sTime : sickTime)
3. for (object sTime : sickTime)
4. for (object sTime < sickTime)

Answer

***Option 1:****Correct. This segment correctly accesses each element and iterates though the ArrayList.*

***Option 2:****Incorrect. The proper keyword to use is for.*

***Option 3:****Incorrect. Object with a lowercase "o" is not the proper way to refer to the Object.*

***Option 4:****Incorrect. You do not need to use the comparison operator in this case but are required to use the colon. Additionally, object with a lowercase "o" is not the proper way to refer to the Object.*

**Correct answer(s):**

1. for (Object sTime : sickTime)

An ArrayList has been created, populated, and manipulated, and loops have been used to process an ArrayList.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a03_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited