Ryan Dougherty

CSE110 Lecture Notes

Version 1.0

To Jennifer

Table of Contents

1 Introduction

Welcome to these lecture notes for CSE110! This guide will help you on your journey in learning the basics of programming in Java, including methodologies for problem solving using computers. Programming is becoming more of an important tool in the last few decades. Understanding of these technological shifts, therefore, is imperative.

The topics are split up according to the Table of Contents above, and their corresponding page numbers. At the end of each chapter are written and programming exercises. The written exercises are for testing your knowledge of the material, and the programming ones are to see if you can write programs using the material from that chapter.

If there are any questions/errors/comments that you want to send regarding the material, send an email to: Ryan.Dougherty [at] asu.edu.

2 Computers

2.1 Motivation

This chapter concerns basic knowledge of computers. In order to write programs for computers, we need to have a basic understanding of how computers work. Once we have this understanding, we can write programs that make use of the computer's hardware and software. In recent years, computers have become much faster due to advancements in hardware speed and software breakthroughs. However, there is only a limit to how fast these advancements can take us. Therefore, we need to write programs that use the hardware efficiently.

2.2 Introduction to Java

Java is an object-oriented programming language invented by Sun Microsystems in 1995, and Sun was acquired by Oracle in 2009. One large benefit of Java is that it is run in a *virtualmachine*, which means that it can be run on almost all major platforms: Windows, Mac OS X, Linux, Solaris, etc.

2.3 Java Program Structures

All Java programs follow the same basic structure. A typical Java file (with a ".java" extension) is composed of 1 or more *classes*, which have 1 or more *methods*, which have 1 or more *statements*. We will give more rigor to what these terms mean later. Java classes are (typically) provided in packages, which contain one 1 or more classes.

2.4 First Java Program

A typical first program is called a "Hello, World" program to the screen. The reason that this is the first program for many programmers is that they can demonstrate that they know the basic structure of a Java program.

To print to the screen, a program (that must be saved in a text file that is called "HelloWorld.java") to do this is:

```
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello, World");
        }
}
```

You must type this program in a raw text editor, not in a rich text editor such as Microsoft Word. The reason for this is that Microsoft Word adds special characters, which are not recognized when you compile and run your program (see below).

For many beginners, this program may be a little daunting and confusing. But as we can see, this program follows the basic structure of a Java program: it has 1 class, named "HelloWorld"; it has one method, named "main", and it has one statement, which prints to the screen.

For each of the terms, the precise meaning of them will be covered in later chapters. However, the basic structure of:

```
public class YourProgram {
        public static void main(String[] args) {
                // Insert statements here
        }
}
```

will be enough for most beginning Java programs. As a programmer, you will put the statements between the opening and closing brace of the main method.

Note: the text "Insert statements here" is in green. This is a "comment", in which you can add any text. There are 3 types of Java comments:

- Line comments - start with `//`, and last for the rest of the line.
- Multiline comments - start with `/*` and end with `*/`, and last until the `*/` is reached.
- JavaDoc comments - these have the same form as multiline comments, but have a purpose in providing documentation for Java code. They are typically put right before a method declaration.

2.5 Running Java Programs

Now that we understand how to create Java programs, we want to run them on our computers. In order to run a Java file, we need to do 2 steps: compile it, and then run it.

Compiling Java Programs A compiler (for most programming languages) is software that translates the user-created code into object (or "machine") code. Obviously, user-created code is readable by the user, since the user created it. However, object code is not readable, since it is code that the computer can read, and then execute.

There are a few ways of compiling a Java file:

- If you have an Integrated Development Environment (IDE), such as Eclipse, Netbeans, etc. on your machine (they are available on the web for free), running your program is done by clicking the green arrow ("Run") button in the toolbar. This operation will compile and run the program.

– If you are not using an IDE, and have a Terminal/Command Prompt, etc. application in which you supply commands, you enter these commands to compile and then run a Java program:
`javac YourProgramName.java`
`java YourProgramName` (Note: no ".java" extension on this command)

However, if you mistyped during creating your program, you encounter what is called a "compilation error", which shows an appropriate error message. Common errors include:

– Incorrect class name - the name of the class must exactly match the name of the file that it is saved in.
– Invalid class name - there are rules for what a class name can be. For example, it cannot be a Java keyword (i.e. public, class; all of the words that highlight in blue).
– No semicolon after statements - all statements must end in a semicolon.
– Not putting quote marks around what is put inside the "System.out.println" statement. This is called a "string" of characters. If no quote marks are put, then the Java compiler cannot recognize what Hello, World means. However, it does know what "Hello, World" means.

2.6 Notes

– There are 2 kinds of print statements in Java:
  • `System.out.print` - prints out exactly what is given in the parentheses.
  • `System.out.println` - same as `print` but adds a newline after the text given in the parentheses. The next time something is printed, it will start on the next line.
– There are numerous special characters that appear in between quote marks:
  • `\n` - a newline character. Suppose we have this line of code:

    ```
    System.out.print("Testing newline\nTest2");
    ```

    The output for this code would be:

    ```
    Testing newline
    Test2
    ```

    Notice that the character `\n` was not printed because it has a special purpose: providing a new line. Therefore, one can see that the following lines of code do the same thing:

    ```
    System.out.println("Text");
    System.out.print("Text\n");
    ```

  • `\t` - a tab character. The text that follows `\t` will appear in the same manner as a tab key is used in any other text document.
  • `\"` - to print a double quote.

2.7 Written Exercises

1. What does a compiler do?
2. Consider the following Java Program:

```
public class VendingMachine {
      public static void main(String[] args) {
            System.out.println("Please insert 25c");
      }
}
```

   By what name would you save this program on your hard disk?
3. Is Java a functional language, procedural language, object-oriented language, or logic language?
4. What is a plain text file?
5. How is a text file different than a .doc file?
6. What is a source program?
7. What is Java bytecode?
8. What is the program that translates Java bytecode instructions into machine-language instructions?
9. Is Java case-sensitive?

2.8 Programming Exercises

1. Write a program to print the following to the screen using a number of print statements:

```
  *
 * *
* * *
```

3 Data Types

3.1 Motivation

Now that we know how to compile and run programs, and print to the screen, we will bump up the complexity of our programs a little. In this chapter, we will be exploring data types, and how to do basic calculations.

3.2 Types

There are 8 "primitive" (fundamental) data types in Java (we will explain constraints that they have later):

- `byte` - an integer with a small range.
- `short` - same as `byte`, but a larger range.
- `int` - same as `short`, but even larger range.
- `long` - same as `int`, but even still larger range.
- `float` - a "real" value, that has some precision.
- `double` - same as `float` but with more precision.
- `boolean` - a type that takes one of two possible values: `true`, or `false`.
- `char` - a Unicode character. Think of `a` or `b` as `char`s.

Now, we give the range for each of the primitive data types (all inclusive on both ends):

- `byte`: -128 to +127.
- `short`: -32768 to +32767.
- `int`: -2^{31} to $+2^{31} - 1$.
- `long`: -2^{63} to $+2^{63} - 1$.
- `float`: 1.4×10^{-45} to 3.4×10^{38}.
- `double`: 4.9×10^{-324} to 1.7×10^{308}.
- `boolean`: `true` or `false`.
- `char`: 0 to 65535.

Now that we know the 8 fundamental data types, we want to know how to use them. To do that, we have to *declare* a *variable* with a type.

3.3 Variable Declaration

Like classes in the previous chapter, declaring variables follow the same basic structure, which is:

$$Type\ VariableName = Value;$$

For example, if we want a `int` variable called `val` to have a value of 2, we would write the line of code as:

```
int val = 2;
```

Also, we can declare several variables of the same type in the same line using a comma between the names:

```
int val = 2, otherVal = 3;
```

Notes:

- In a list of declarations, there is no need to declare the type twice.
- The convention in Java names is to follow camel casing: the first "word" is lowercase, and the subsequent "words" in the same variable are capitalized (see example above).

Now, there are only certain names that our Java variables can have. The rules are:

- The first character must be a letter (either upper or lowercase), the underscore symbol, or a dollar sign ("$"). However, a variable name cannot start with a number.
- Any character other than the first may be letters, underscores, the dollar sign, or also numbers.

Here are some variable declarations using all primitive types:

```
byte b = 1;
short s = 2;
int i = 300;
long l = 1000L; // note the 'L'
float f = 0.56f; // note the 'f'
double d = 523.6; // no suffix
boolean b = true;
char c = 'a'; // chars are surrounded in single quotes
```

After we have initialized a variable, we can also modify it without having to declare it again. Here is an example:

```
int i = 300;
i = 301; // modifies i's value to be 301
// int i = 301; // this causes a compiler error
                //(declare once)
```

Now that we know how to declare and modify values, we need to learn how to do various operations on them, such as addition, subtraction, multiplication, and division. An example showing all of them is the following (with comments):

```
/* Addition */
// int works the same as byte and short
int i1 = 0, i2 = 3;
int i3 = i1 + i2; // adds i1's value (0) and i2's (3)

// doubles and floats work the same way:
double d1 = 2.5, d2 = 3.0;
```

```
float d3 = d1 + d2;
float f1 = 2.5f, f2 = 3.0f;
float f3 = f1 + f2;

// Can add across types, but is promoted to "most
   precise"
// If we add double + int, the result is a double
int i4 = 4;
double d4 = 5.3;
double d5 = d4 + i4;

// cannot do operations on booleans (compiler error)
// boolean b1 = true + false;

/* Subtraction */
// Works with primitive types the same way as addition
int i5 = 0, i6 = 3;
int i7 = i5 - i6; // i6 has value -3

/* Multiplication */
// Multiplication is the same as addition in the same
   type
// However, we promote to "most precise" value if
   multiple types
int i8 = 5;
double d6 = 5.3;
double d7 = d6*i8; // promote i8 to a double valued 5.0

/* Division */
// Division is easy to understand except for when the
   numerator and denominator are both ints.
int i9 = 10, i10 = 3;
// What is i9 / i10? We expect 3.33333..., but we get 3
    instead. Java does "integer division", where the
   numerator and denominator are both ints, which is
   truncating all the fractional parts.
int i11 = i9 / i10; // i11 has value 3.
// However, if we "cast" one of the values to a double,
    then we do not have this problem
double d8 = (double) i9 / i10; // cast i9 to a double
   of value 10.0
// d8 now has value 3.333333...
```

Another operation is called the mod operator (or “remainder”):

```
int i1 = 10, i2 = 3;
```

```
int i3 = i1 % i2; // i3 has value 1, since 10 remainder
    3 = 1.
```

We can also "shorten" times when we are modifying the same value with another. For example:

```
int i = 5;
i += 3; // equivalent to: i = i + 3;
i -= 3; // equivalent to: i = i - 3;
i *= 3; // equivalent to: i = i * 3;
i /= 3; // equivalent to: i = i / 3;
```

For `int` variables, there are 2 operations that are done very often, so they were put into the language: increment and decrement:

```
int i = 2;
i++; // equivalent to: i += 1;
i--; // equivalent to: i -= 1;
```

3.4 String

Now that we fully understand how primitive data types work, we can move on to using classes provided by the Java API (Application Programmer Interface). One such class is provided in the Java package `java.lang`, called String. This package is so often that Java automatically imports it for you, so you do not have to do extra work.

A String is a set of characters in between double quotes. Any character can be inside a string, except for a double quote. Examples are:

```
String str1 = "I love";
String str2 = "Java";
String str3 = str1 + str2; // is now "I loveJava"
// str3 is called the "concatenation" of str1 and str2
int i1 = 3;
str3 = str1 + i1; // has value "I love3"
```

However, there is another way to declare a String variable, and that is done using the Java keyword `new`:

```
String str1 = new String("I love"); // same as above
```

String is the only Java class that can declare (also called "instantiate") a variable without the `new` keyword. All other Java classes must be instantiated with this keyword.

Now we need to see what we can do with these String variables. As we covered in the last chapter, every Java class has one or more methods. The syntax for calling a method is:

$$variable.methodname(optionalparameters)$$

Examples of various methods in Java for the String class are:

```
String str1 = "I love Java";
/*String.toUpperCase() and toLowerCase*/
String upper = str1.toUpperCase(); // "I LOVE JAVA"
String lower = str1.toLowerCase(); // "i love java"

/*String.replace(char c, char d)*/
// Replaces all instances of c with d
String repl = str1.replace('a', 'p'); // "I love Jpvp"

// We can even chain method calls
String myst = str1.replace('a', 'p').toUpperCase(); //
   "I LOVE JPVP"

/*String.length() - int which is # of characters*/
int len = str1.length(); // 11, since there are 11
   characters

/*String.charAt(int n) - gets the nth character*/
// Strings are 0-indexed, so the first character is at
   index 0
// Error at runtime if input >= length()
char firstChar = str1.charAt(0); // has value 'I'
char secondChar = str1.charAt(1); // has value ' '
char lastChar = str1.charAt(str.length()-1); // 'a'

/*String.substring(int n, int m)*/
// gives a String that consists of index n through m,
   not inclusive on m
String substr1 = str1.substring(0, 3); // "I lo"
String substr2 = str1.substring(2, 6); // "love"

/*String.indexOf(char c)*/
// Returns index (int) of the first instance of c;
   otherwise, -1
int found1 = str1.indexOf('o'); // 4
int found2 = str1.indexOf('f'); // -1
```

3.5 Scanner

Now that we know how to modify String variables, it is important to know how to get input from the user so that we can make our programs usable. For that, we use the Scanner class in the `java.util` package.

However, since this class is not `java.lang`, we must *import* it from Java, by putting an import statement at the beginning of our file:

```
import java.util.Scanner; // import the Scanner class
   from the java.util package
// Equivalent: import java.util.*; // import everything
    from this package
public class SomeClass {
        // ...
}
```

So how do we use this class? We must initialize it with the `new` keyword:

```
Scanner s = new Scanner(System.in); // System.in means
   from the keyboard
```

This code creates an variable (often called "object" in this case, since it is a class) in which we can call various methods on it to get input from the user.

3.6 Written Exercises

1. Give the output of the following program:

```
public class Example {
     public static void main(String[] args) {
          int y = 2, z = 1;
          z = y * 2;
          System.out.print(y + z);
     }
}
```

2. Consider the following program:

```
public class Example {
     public static void main(String[] args) {
          String str = new String("Arizona state
             university");
          char ch1 = str.toLowerCase().toUpperCase()
             .charAt(0);
          char ch2 = str.toUpperCase().charAt(8);
          char ch3 = str.toUpperCase().charAt(str.
             length() - 1);
          System.out.println("character 1 is: " +
             ch1);
          System.out.println("character 2 is: " +
             ch2);
          System.out.println("character 3 is: " +
             ch3);
```

```
        }
}
```

What will be the output?

3. Consider the following program:

```
public class Example {
        public static void main(String[] args) {
                int num1 = 4, num2 = 5;
                System.out.println("4" + "5");
                System.out.println(num1 + num2);
                System.out.println("num1" + "num2");
                System.out.println(4+5);
        }
}
```

What will be the output?

4. Which of the following invokes the method length() of the object str and stores the result in val of type int?

```
a) int val = str.length();
b) int val = length.str();
c) int val = length().str;
d) int val = length(str);
```

5. Evaluate each of the following expressions.

```
String s = "Programming is Fun";
String t = "Workshop is cool";
a) System.out.println(s.charAt(0) + t.substring(3,
   4));
b) System.out.println(t.substring(7));
```

6. Evaluate each of the following expressions.

```
int j = 11;
int k = 3;
String s = "Ford Rivers";
a) j / k
b) j % k
c) s.substring(1, 5)
d) s.length()
e) s.charAt(3)
```

7. True or False? The type String is a primitive data type.
8. True or False? The type String is a primitive data type.
9. Write the output of the following program:

```
public class Question {
        public static void main(String[] args) {
                String str = "hello";
```

```
            System.out.println("abcdef".substring(1,
               3));
            System.out.println("pizza".length());
            System.out.println(str.replace('h', 'm'));
            System.out.println("hamburger".substring
               (0, 3));
            System.out.println(str.charAt(1));
            System.out.println(str.equals("hello"));
            System.out.println("pizza".toUpperCase());
            System.out.println(Math.pow(2, 4));
            double num4 = Math.sqrt(16);
            System.out.println(num4);
      }
}
```

10. Write the output of the following program:

```
public class Question {
      public static void main(String[] args) {
            String s1 = new String("Clinton, Hillary")
               ;
            String s2 = new String("Obama, Barack");
            System.out.println(s1.charAt(2));
            System.out.println(s1.charAt(s1.length() -
                1));
            System.out.println(s2.toUpperCase());
            System.out.println(s2.substring(
                 s2.indexOf(",") + 2, s2.length());
      }
}
```

11. What value is contained in the integer variable length after the following statements are executed?

```
length = 5;
length += 3;
length = length * 2;
```

12. What is the result of 2/4 when evaluated in Java? Why?

3.7 Programming Exercises

1. Write a Java program that asks the user for the radius of a circle and finds the area of the circle.
2. Write a Java program that prompts the user to enter 2 integers. Print the smaller of the 2 integers.

4 Decisions

4.1 Written Exercises

1. What is the output of the following code?

```
int depth = 8;
if (depth >= 8) {
     System.out.print("Danger: ");
     System.out.print("deep water. ");
}
System.out.println("No swimming allowed.");
```

2. What is the output of the following code?

```
int depth = 12;
int temp = 42;
System.out.print("The water is: ");
if (depth >= 8)
     System.out.print("deep ");
if (temp <= 50 && depth <= 12)
     System.out.print("cold ");
System.out.println(" wet.");
```

3. If k holds a value of the type int, then the value of the expression:

```
k <= 10 || k > 10
```

   a) must be true
   b) must be false
   c) could be either true or false
   d) is a value of type int

4. Consider the following code:

```
String str1 = "Java is fun";
String str2 = "Java is fun";
if ( /* */ )
     System.out.println("String1 and String2 are the
         same");
else
     System.out.println("String1 and String2 are
        different");
```

   Fill in the missing condition to check if str1 and str2 are the same.

5. Evaluate the following expressions, assuming that x = -2 and y = 3.

```
a) x <= y
b) (x < 0) || (y < 0)
c) (x <= y) && (x < 0)
d) ((x + y) > 0) && !(y > 0)
```

6. Write the output of the following code:

```
int grade = 45;
if (grade >= 70)
     System.out.println("passing");
if (grade < 70)
     System.out.println("dubious");
if (grade < 60)
     System.out.println("failing");
```

7. Write the output of the following code:

```
String option = "A";
if (option.equals("A"))
     System.out.println("addRecord");
if (option.compareTo("A") == 0)
     System.out.println("deleteRecord");
```

8. Write the output of the following code:

```
double x = -1.5;
if (x < -1.0)
     System.out.println("true");
else
     System.out.println("false");
     System.out.println("after if...else");
```

9. Write the output of the following code:

```
int j = 8;
double x = -1.5;
if (x >= j)
     System.out.println("x is high");
else
     System.out.println("x is low");
```

10. Write the output of the following code:

```
double x = -1.5;
if (x <= 0.0) {
     if (x < 0.0)
          System.out.println("neg");
     else
          System.out.println("zero");
}
else
     System.out.println("pos");
```

4.2 Programming Exercises

1. Write a program that asks for 3 integers and prints the median value of the three integers.
2. Write code that ensures that an int variable called number is an odd integer.

5 Loops

5.1 Written Exercises

1. What are the 3 kinds of loops in Java?
2. What is the output of the following loop? How many times does the loop execute?

```
int n = 979;
for (int j = 0; j <= n; j++) {
      System.out.println("Hello");
}
```

3. What is the output of the following loop? How many times does the loop execute?

```
int j = 1;
int n = 5;
while (j <= n) {
      System.out.println("Hello");
      n--;
}
```

4. What is the output of the following loop? How many times does the loop execute?

```
int n = 5;
for (int j = 1; j <= n; j += 3) {
      System.out.print("Hello ");
      int k = j;
      while (k < n) {
            System.out.println("Good Morning");
            k++;
      }
      j--;
}
```

5. What is the output of the following code?

```
String name = "Richard M. Nixon";
boolean startWord = true;
for (int i = 0; i < name.length(); i++) {
      if (startWord)
            System.out.println(name.charAt(i));
      if (name.charAt(i) == ' ')
            startWord = true;
      else
            startWord = false;
}
```

6. What is the output of the following loop? How many times does the loop execute?

```
int j = 1;
while (j <= 11) {
      System.out.println("Hello");
      j = j + 3;
}
```

7. What is the output of the following code?

```
int n = 1, i = 1;
while (i < 7) {
      n = n * i;
      i += 2;
}
System.out.print(n);
```

5.2 Programming Exercises

1. Write a loop that reads in int values until the user enters 0 and prints out how many values entered are greater than 10.
2. Write a loop that will print out every other letter in a String str. For example, if the String was "Hello There", then "HloTee" will be printed.

6 Introduction to Classes

6.1 Written Exercises

1. Which of the following enforces Encapsulation?
   a) Make instance variables private
   b) Make methods public
   c) Make the class final
   d) Both a and b
   e) All of the above
2. Use the following class to answer the questions below:

```
public class Store {
        private int quantity;
        private double price;

        public Store(int q, double p) {
                quantity = q;
                price = p;
        }

        public int getQuantity() {
                return quantity;
        }

        public void setPrice(double p) {
                price = p;
        }

        public double calcTotal() {
                return price * quantity;
        }
}
```

   a) What is the name of the class?
   b) List all instance variables of the class.
   c) List all methods of the class.
   d) List all mutators in the class.
   e) List all accessors in the class.
   f) List which method is the constructor.
3. True or False? If no constructor is provided, then Java automatically provides a default constructor.
4. True or False? A method must have at least 1 return statement.

6.2 Programming Exercises

1. For the Store class in the Written Exercises above, do the following:
   a) Write a mutator for the quantity.
   b) Write an accessor for the price.
   c) Write a line of code that will create an instance called videoStore that has quantity 100 and a price of 5.99.
   d) Call the calcTotal method with the videoStore object (from part c) to print out the total.
2. Correct the following class definition if you think it will not work:

```
public class Student {
      private String name, major;
      public Student() {
            name = "???";
            major = "xxx";
      }
      public Student(String n, String m) {
            n = name;
            m = major;
      }
      public String getMajor() {
            return m;
      }
      public String getName() {
            return n;
      }
}
```

3. Implement a class called AsuStudent. The class should keep track of the student's name, number of classes registered, hours spent per week for a class (consider a student devotes the same amount of time for each of his/her classes per week). Implement a toString method to show the name and number of classes registered by a student, a getName method to return the name of the student, a getTotalHours method to return the total number of hours per week, and a setHours method to set the number of hours the student devotes for each class.

7 Methods

7.1 Written Exercises

1. Write the output generated by the following program:

```
public class Two {
     private double real, imag;
     public Two(double initReal, double initImag) {
          real = initReal;
          imag = initImag;
     }
     public double getReal() {
          return real;
     }
     public double getImag() {
          return imag;
     }
     public Two mystery(Two rhs) {
          Two temp = new Two(getReal()+rhs.getReal()
              ,
                getImag()+rhs.getImag());
          return temp;
     }
}
public class Test {
     public static void main(String[] args) {
          Two a = new Two(1.2, 3.4);
          Two b = a.mystery(a);
          Two c = b.mystery(b);
          System.out.println("1. " + a.getReal());
          System.out.println("2. " + a.getImag());
          System.out.println("3. " + b.getReal());
          System.out.println("4. " + b.getImag());
          System.out.println("5. " + c.getImag());
     }
}
```

2. Using these 2 classes, write the output of the following program:

```
public class CDPlayer {
     private int totalTime;
     public CDPlayer() {
          totalTime = 0;
     }
     public int totalPlayTime() {
          return totalTime;
     }
```

```
        public void play(CDTrack aTrack) {
            totalTime += aTrack.getPlayTime();
        }
    }
    public class CDTrack {
        private String myTitle;
        private int myPlayTime, myTimesPlayed;
        public CDTrack(String trackTitle, int playTime)
            {
            myTitle = trackTitle;
            myPlayTime = playTime;
            myTimesPlayed = 0;
        }
        public int getPlayTime() {
            return myPlayTime;
        }
        public void wasPlayed() {
            myTimesPlayed++;
        }
        public String toString() {
            String result = "";
            int minutes = myPlayTime / 60;
            int seconds = myPlayTime % 60;
            result += myTitle + " " + minutes + ":" +
                seconds;
            result += " #plays = " + myTimesPlayed;
            return result;
        }
    }
    public class RunCDPlayer {
        public static void main(String[] args) {
            CDTrack t1 = new CDTrack("Day Tripper",
                150);
            CDTrack t2 = new CDTrack("We Can Work it
                Out", 200);
            CDTrack t3 = new CDTrack("Paperback Writer
                ", 138);
            CDPlayer diskPlayer = new CDPlayer();
            t1.wasPlayed();
            diskPlayer.play(t1);
            t2.wasPlayed();
            diskPlayer.play(t2);
            t1.wasPlayed();
            diskPlayer.play(t1);
            System.out.println(t1.toString());
```

```
            System.out.println(t2.toString());
            System.out.println(t3.toString());
            System.out.println("Total play time: " +
                  (diskPlayer.totalPlayTime() / 60) + "
                     :" +
                  (diskPlayer.totalPlayTime() % 60));
        }
}
```

7.2 Programming Exercises

1. Write a boolean method called allDifferent that takes 3 int numbers and returns true if the numbers are all different, and false otherwise.
2. Write a boolean method called isPrime that takes in an int number, and returns true if the number is prime, and false otherwise.

8 Static

8.1 Written Exercises

1. What is a static variable? What is a static method?
2. Using the code below, how many copies of the variable number exist after instantiating 374 different AmazingClass objects?

```
public class AmazingClass {
      private static int number;
      public AmazingClass(int a) {
            number = a;
      }
      public int twice() {
            number *= 2;
            return number;
      }
}
```

3. Using the code from above, what is the value of number after each of the following statements? (For each part, assume the preceding parts have already been executed).

```
AmazingClass ac1 = new AmazingClass(3);
AmazingClass ac2 = new AmazingClass(7);
ac1.twice();
ac2.twice();
```

8.2 Programming Exercises

9 Method Overloading

9.1 Written Exercises

1. What is method overloading?
2. What are the valid method headings assuming they are written in the same class?

```
a) public void Void()
b) public double void f2()
c) public double sum(int left, right)
d) public String string(int n)
e) public BankAccount bankAccount()
```

9.2 Programming Exercises

10 Arrays

10.1 Written Exercises

1. What are the indices for the first and last positions of any array?
2. Immediately after instantiating a new array of primitives (ints, doubles, etc.), what fills the array? What about an array of objects?
3. What happens when you try to access an array element past the end of the array?
4. Use the following array x to answer the following questions:

4 8 5 1 6 3 2

   a) What value is given by x[1]?
   b) What value is given by x[6]?
   c) What value is given by x[7]?
   d) What value is given by x.length?

10.2 Programming Exercises

1. Instantiate three arrays called x, y, and z of type int, String, and BackAccount (respectively), all of size 10.
2. Write a for-loop to sum all of the elements of an array x of type int.
3. Write a for-loop to double each element in an array x of type int.
4. Write code to store the largest number in an int array x into a variable called max.
5. Write code to count how many numbers in the array are strictly larger than 4, and store that total in a variable called total.
6. Write code to print out every other element in an array separated by tabs.
7. Write code to shift each number one place to the right (Note: there will be 2 copies of the 1st element when the code finishes).
8. Write code to print the contents of an array in reverse order, one element for each line.
9. Write a method called append that appends the two arrays passed as arguments and returns an array of type int as the result. For example, if the first array argument was {1, 2, 3}, and the second was {4, 5, 6, 7}, append returns {1, 2, 3, 4, 5, 6, 7}.
10. Write a method called findMin that returns the smallest element in an array that is passed as an argument. For example, if the array was {4, 7, 9, 12, 8, 1, 5}, the method would return 1.

11 Searching

11.1 Written Exercises

1. Use the sorted list below and use binary search to look for Mike in the list. Show all the names that will be compared before Mike is found. Then, repeat the same process for Cathy (note: Cathy is not in the list).

Aaron Betsy Doug Elise Mike Pat Steven

2. What is the benefit of using binary search over linear search?

11.2 Programming Exercises

12 Sorting

12.1 Written Exercises

1. Write the contents after each step of selection sort and insertion sort (assume by alphabetical order).

Mike Betsy Aaron Steven Doug Pat Elise

12.2 Programming Exercises