HOMEWORK PROBLEMS 10

1) What happens if you delete the first `B` constructor in `C10h1.java` (a copy of `Inheritance2` in Fig. 10.4). Will the program compile without error. If so, what does it display?

2) Suppose `B` is a subclass of `A`. Suppose `A` has a `public` method `f` with no parameters that displays `hello`. Suppose `B` has a `public` method `f` with one `int` parameter that displays the value of its parameter. Thus, `B` has two `f` methods, one of which is inherited. Which `f` methods are called when the following code is executed:

```
B b = new B();
b.f();
b.f(5);

A a;
a = b;
a.f();
```

Why is the inherited `f` not overridden by the `f` defined in `B`? Is the following call legal:

```
  a.f(5);
```

3) Suppose `B` is a subclass of `A`. Is an `A` object also a `B` object?

4) What does the default constructor do?

5) Why can the `toString` method be called via any non-null reference?

6) Run a test program to confirm that `Object` has a `hashCode` method that takes no arguments and returns an `int`. Because `Object` has a `hashCode` method as well as an `equals` and a `toString` method, *every* class inherits these three methods. Classes can, of course, override the methods inherited from `Object`. `hashCode` is method that can be used for hash coding (see homework problem 38 in Chapter 9).

7) What is returned by the two calls of `hashCode` in the following code:

```
Integer i= new Integer(12345);
System.out.println(i.hashCode());
Double d = new Double(123.45);
System.out.println(d.hashCode());
```

8) Create a class `Cub` that does not define a `hashCode` method or a `toString` method. Execute the following code:

```
Cub c = new Cub();
int i = c.hashCode();
System.out.println(Integer.toHexString(i));
System.out.println(c.toString);
```

The call of `toHexString` method above converts the integer in `i` to hex (i.e., base 16). In hex, the symbols A, B, C, D, E, and F have values equal to the decimal numbers 10, 11, 12, 13, 14, and 15, respectively. Where do the `hashCode` and `toString` methods called in the code above come from? How is the string returned by the `toString` method constructed?

9) What is wrong with the following program? Compile it to check your answer. Modify the program so it works.

```
class C10h9
{
   public static void main(String[] args)
   {
       Bud b = new Bud();
       b.display();
   }
}
//================================================
class Ale
{
   protected int x = 1;
   public Ale(int x)
   {
      this.x = x;
   }
}
//================================================
class Bud extends Ale
{
   private int y = 2;
   public void display()
   {
      System.out.println("x = " + x + " y = " + y);
   }
}
```

10) What is displayed by the following program? Run it to check your answer. What can you conclude about the order in which constructors are executed?

```
class C10h10
{
   public static void main(String[] args)
   {
      Pie c = new Pie();
   }
}
//================================================
class Ate
{
   public Ate()
   {
      System.out.println("Ate constructor");
   }
}
//================================================
class My extends Ate
{
   public My()
   {
      System.out.println("My  constructor");
   }
}
```

```
//================================================
class Pie extends My
{
   public Pie()
   {
      System.out.println("Pie constructor");
   }
}
```

11) Create a class `Asp` and a subclass `Bee`. `Asp` should have a `public int` field `x` whose initial value is 1, and a `public` method `f` that displays "`in Asp`". `Bee` should have a `public int` field `x` whose initial value is 2, and a `public` method `f` that displays "`in Bee`". Thus, a `Bee` object has two `f` methods and two `x` instance variables. Which `x` variables are accessed and which `f` methods are called in the following sequence:

```
Asp a = new Bee();              // a points "down" to Bee object
a.f();                          // Which f() is executed?
System.out.println(a.x);        // Which x is displayed?
Bee b = (Bee)a;                 // b points "across to Bee object
b.f();                          // Which f is executed?
System.out.println(b.x);        // Which x is displayed?
((Bee)a).f();                   // Which f() is executed?
System.out.println(((Bee)a).x); // Which x is displayed?
```

The method executed depends on the *type of the object*—not the type of the reference. Because both `a` and `b` point to a `Bee` object, all the calls of `f` in the code above call the `f` method defined in the `Bee` class. However, the instance variable accessed depends on the *type of the reference*—not on the type of the object. `x` defined in `Bee` does *not* override the inherited `x`. Instead, the `x` defined in `Bee` **shadows** the inherited `x`.

12) Create a `Coy` class that contains a private `int x`, a constructor that initializes `x` to 5, and a method `f` that displays `x`. It does not contain its own `toString` or `equals` methods. Then execute in `main`

```
Coy r, s;
r = new Coy();
s = new Coy();

if (r == s)
   System.out.println("equal");
else
   System.out.println("not equal");
```

What happens? Why? Then execute

```
if (r.equals(s))
   System.out.println("equal");
else
   System.out.println("not equal");
```

Where is this `equals` method coming from? What happens? Why? Then execute

```
System.out.println(r.toString());
System.out.println(s.toString());
```

What happens? Why? Where is the `toString` method coming from?

13) Write a program that has three classes: Ark, Boa, and C10e8. Boa is a subclass of Ark. Ark has a public method f that displays hello. Boa has a public method f that first calls the f method in Ark and then displays bye. Both f methods have no parameters. C10e8 contains main. main should execute

```
Ark a = new Ark();
a.f();
Boa b = new Boa();
b.f();
a = b;
a.f();
```

Before you run your program, predict what it will display.

14) Create three classes: Ash, Bow, and Cup. Cup is a subclass of Bow; Bow is a subclass of Ash. Ash should have a private int x field; Bow should have a private int y field; Cup should have a private int z field. Each class should have a constructor with one parameter that provides the initial value for the data field defined in that class. The Cup constructor should pass 2 to the Bow constructor. The Bow constructor should pass 1 to the Ash constructor. Each class should have its own public display method. The display method in Ash should display x. The display method in Bow should display x and y. It should access x using a public accessor method xGet in Ash. The display method in Cup should display x, y, and z. It should access x and y using public accessor methods xGet in Ash and yGet in Bow. Create a C10e9 class with a main method that executes

```
Ash a = new Ash(10)
a.display();
Bow b = new Bow(20);
b.display()
Cup c = new Cup (30);
c.display();
```

15) Recall that println is an overloaded method. The implementation of the version that has a parameter of type Object is

```
public void println(Object obj)
{
   println(obj.toString());
}
```

The obj parameter in the println method above can be passed the reference to any type of object because a reference of type Object is a “universal” pointer. Why is it safe for this println method to call the toString method in the obj object? Does every object have a toString method? What other methods can be called safely?

16) Run the code below (it is in `C10h16.java`). From the output displayed, you will see that the `remove` method works differently depending on the base type of the `ArrayList`. Why?

```
import java.util.ArrayList;
class C10h16
{
   public static void main(String[] args)
   {
      ArrayList<Integer> ial = new ArrayList<Integer>();
      Integer i1 = new Integer(1);
      Integer i2 = new Integer(1);    // construct identical object
      if (i1 == i2)
         System.out.println("i1 == i2 is true");
      else
         System.out.println("i1 == i2 is false");
      ial.add(i1);
      ial.remove(i1);                 // remove what was just added
      System.out.println(ial.size()); // should be zero
      ial.add(i1);
      ial.remove(i2);                 // will passing i2 remove i1?
      System.out.println(ial.size()); // is it zero?

      ArrayList<Strange> ral = new ArrayList<Strange>();
      Strange r1 = new Strange();
      Strange r2 = new Strange();     // construct identical object
      if (r1 == r2)
         System.out.println("r1 == r2 is true");
      else
         System.out.println("r1 == r2 is false");
      ral.add(r1);
      ral.remove(r1);                 // remove what was just added
      System.out.println(ral.size()); // should be zero
      ral.add(r1);
      ral.remove(r2);                 // will passing r2 remove r1?
      System.out.println(ral.size()); // is it zero?
   }
}
//==================================================================
class Strange
{
   int x = 3;
}
```

17) `javadoc` is a program that comes with the Java compiler and interpreter. It automatically creates a documentation file on any *public* class. It extracts out information on the public members in a public class and creates a file with this information that is viewable with a web browser. If the class includes `javadoc` **comments** (comments that start with /** and end with */ that appear immediately before the public members), these comments will also be included in the file created. `javadoc` comments can include **tags** that provide specific information on a public member. For example, the `@param` tag provides information on parameters; the `@return` tag provides information on the value returned by a method. For example, in the class below, the `@param` provides information on the parameters of the constructor; `@return` provides information on the return value of the `toString` method.

```
/** This class models a rational number (a number
    that can be represented as the ratio of two integers).
*/
public class RationalNumber    // class must be public for javadoc
{
   private int numerator, denominator;
   /**
      @param n numerator of rational number
      @param d denominator of rational number
   */
   public RationalNumber(int n, int d)
   {}
   /** adds the rational number r to this rational number */
   public RationalNumber add(RationalNumber r)
   {}
   /** add the rational number r to this rational number */
   public RationalNumber multiply(RationalNumber r)
   {}
   /** reduces this rational number to lowest terms */
   public void reduce()
   {}
   /**
       @return  a string with rationa number in
       "numerator/denominator" form
   */
   public String toString()
   {}
}
```

Run javadoc on the RationalNumber.java file by entering

```
javadoc RationalNumber.java
```

With a web browser, view the file created and compare with RationalNumber.java.

18) Create three public classes (they must be in separate files because they are public) : CollegeMember, Student, and Professor. Student and Professor are subclasses of CollegeMember. CollegeMember has a String name field and a String telNumber field. Its constructor has two parameters that provide the initial values for the name and telNumber fields. CollegeMember also has accessor methods getName and getTelNumber that return name and telNumber, respectively. Student has an int year field (1 = first year, 2 = second year, 3 = third year, 4 = fourth year). Its constructor has three parameters: name, year, and telNumber that provide initial values for the name, year, and telNumber fields. It also has an accessor method getYear that returns year. Professor has an int rank field (1 = assistant, 2 = associate, 3 = full). Its constructor has three parameters: name, rank, and telNumber that provide initial values for the name, rank, and telNumber fields. It also has an accessor method getRank that returns rank. Create a C10h18 class with a main method that executes

```
Student s = new Student("Bert", 2, "555-5555");
System.out.println("name = " + s.getName());
System.out.println("year = " + s.getYear());
System.out.println("telephone = " + s.getTelNumber());
Professor p = new Professor("Jane", 1, "555-9999");
System.out.println("name = " + p.getName());
System.out.println("rank = " + p.getRank());
System.out.println("telephone = " + p.getTelNumber());
```

Include `javadoc` comments in your class files. Use `javadoc` to create documentation files (see homework program 17).

19) Same as homework problem 18 except use only two classes: `Student` and `Professor`, neither of which is a subclass of the other or of `CollegeMember`. `Student` should have a `name`, `year`, and `telNumber` fields, and accessor methods for each of these fields. Its constructor should have three parameters: `name`, `year`, and `telNumber`. `Professor` should have a `name`, `rank`, and `telNumber` fields, and accessor methods for each of these fields. Its constructor should have three parameters: `name`, `rank`, and `telNumber`. Compare your program with your program from homework problem 18. Which is better?

20) Same as homework problem 18 except add a `String address` field to the `CollegeMember` class. Change the constructor of `CollegeMember` so that it also has an `address` parameter. `CollegeMember` should also have an accessor method `getAddress` that returns `address`. In `main`, execute the following code:

```
Student s = new Student("Jane", 2, "555-5555", "5 Bouton");
System.out.println("name = " + s.getName());
System.out.println("year = " + s.getYear());
System.out.println("telephone = " + s.getTelNumber());
System.out.println("address = " + s.getAddress());
Professor p = new Professor("Bert", 1, "555-9999", "3 Oak St.");
System.out.println("name = " + p.getName());
System.out.println("rank = " + p.getRank());
System.out.println("telephone = " + p.getTelNumber());
System.out.println("address = " + p.getAddress());
```

21) Same as homework problem 19 but add a `String address` field to `Student` and `Professor` classes. Modify their constructors accordingly. Add an accessor method `getAddress` to both classes. In `main`, execute the code shown in homework problem 20. Which modification—the modification in homework problem 20 or the modification in this problem—is easier to implement?

22) An **enumeration** creates a new type and associates constants with that type. For example, the following enumeration creates the type `ProfRank` with constants `Assistant`, `Associate`, and `Full:`

```
enum ProfRank {Assistant, Associate, Full}
```

We can then declare a variable of type `ProfRank` and assign it any of the constants listed. For example:

```
ProfRank rank;
rank = ProfRank.Associate;
```

Do homework problem 18 using enumerations rather than integer codes for the student year and professor rank.

23) Write a program that displays in decimal the codes used to represent the space, the digits, the uppercase letters, and the lowercase letters. *Hint*: Use `charAt` on a string that contains the characters whose codes you want to display.

24) Determine the hash code (see homework problems 6, 7, and 8) returned by the `hashCode` method in a `String` object that contains `"AB"`. Repeat for `"BA"`. Are the hash codes different? Because the two strings are not the same, they ideally should hash to different hash codes. Suppose the `hashCode` method computed the hash code simply by summing the individual codes of each letter in the string. Then the hash codes for `"AB"` and `"BA"` would be the same. Simply summing the component data in an object to compute a hash code is a poor technique for hash coding. The hash code should *depend on the position of the component data in an object as well as the data itself*, in which case rearrangements of data would likely produce different hash codes.

25) Same as homework problem 24 but for a class with two integer variables. Use the `hashCode` method inherited from `Object`. Is the hash code returned dependent on the position of the data as well as the data itself? That is, is the hash code returned when the two variables have the values 1 and 2 the same when the two variables have their values switched?

26) The `equals` and `toString` methods in `Object` do not perform functions you would likely need in a program. Why then are they in `Object`? The `hashCode` method is in `Object`. Does this `hashCode` method perform a function you might need in a program? Why is `hashCode` in `Object`?

27) The `Object` class has a `clone` method that duplicates objects. Thus, through inheritance, all objects have a `clone` method. If `r` is a reference to an object, then `r.clone()` returns a reference to a duplicate of the `r` object. Experiment with `clone` to determine if it creates a shallow or deep copy.

28) Implement the `println` method in the `PrintStream` class in Fig. 8.13 that has the parameter of type `Object`.

29) When the program in `C10h29.java` below is executed, it displays

```
A@2f92e0f4
abc
```

The first `println` (line 6) does not display the `x` data in the object. Instead, it displays the class name followed by the `@` sign followed by what looks like the hex address of the object. The second `println` (line 8) correctly displays the contents (`abc`) of the `char` array. Why this inconsistent behavior? *Hint*: See homework problem 28. What does `println` display if passed an `Integer` object? A `Boolean` object? A `Double` object?

What does `println` display for an `int` array? For an `Integer` array?

```
 1 class C10h29
 2 {
 3    public static void main(String[] args)
 4    {
 5       A a = new A();
 6       System.out.println(a);              // x data not displayed
 7       char[] ca = {'a', 'b', 'c'};
 8       System.out.println(ca);             // array displayed
 9    }
10 }
11 //========================================
12 class A
13 {
14    public int x = 1;
15 }
```

30) Write a program in which your main method creates 100 Z objects and then display the static int variable x in class Z. What is the final value of x? What would be the final value of x displayed if x were an instance variable. Assume class Z is

```
class Z
{
   static int x= 10;
   public Z()
   {
      x++;
      System.out.println(x);
   }
}
```