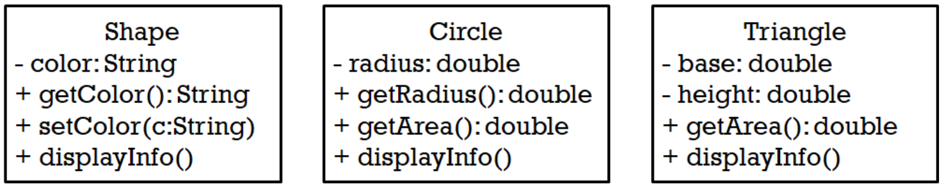
Extra exercises – Week 4

Object Oriented Design

Questions

For some of the questions below, we did not give you “rules” for answering them. We would like you to answer such questions by thinking how it *should* be, based on the rules and concepts that that you have seen.

1. Create the three classes described by the following UML diagrams



2. To the above classes, add appropriate constructors.

3. A method signature is only defined by method name and parameter types. This implies that we cannot override (or overload) a method by having different return type only. This makes sense. Why? What problem would arise if Java allowed it?

4. What is the output when you run the `Test` class below ?

```
class A {
    String s;
    A(){s = "default string"; }   // constructor
    public String toString(){
        return "toString() from A returns: " + s;
   }
}

class B extends A {
   public String toString(){
     return "toString() from B returns: " + s;
   }
}
```

```
class Test {
   public static void main(String[] args){
      B   b  =  new B();
      A   a  =  b;
      System.out.println( a.toString() );
      System.out.println( b.toString() );
   }
}
```

5. Which (if any) of the instructions `(c)-(g)` of `Test` generate compiler errors? [There are no instructions `(a),(b)` because I wanted to avoid confusing with variables `a,b`.]

   What is the output of the program after removing any lines that cause compiler errors?

```
public class A {
   public int n = 3;
   A( ){ System.out.println("A");} // constructor
   public void  foo() { System.out.println( n ); }
}

public class B extends A{
   public  int     n;

   B() { }                                  //  constructor
   B(int n){                                //  constructor
      System.out.println( "B" );
      this.n = n;
   }
   int  foo(int n) {
           System.out.println(this.n) ;
           return n;
   }
}

public class Test {

    public static void main(String[] args) {
       int n = 2;
       A   a = new A();                      //         (c)
       a.foo();                              //         (d)
       a.foo( 7 );                           //         (e)
       B   b  =  new B( 11 );                //         (f)
       n = b.foo( 4 );                       //         (g)
    }
}
```

6. Suppose the classes below are all in the same package.

```
public class Sharer{
   int    sum;
   String  ID;
   Sharer other;

   public Sharer(String ID, int sum){ this.ID = ID;  this.sum = sum; }

   void share(int n){ other.sum += n/2; this.sum += n - n/2;  }  // give half, keep half

   public String toString(){ return ID + " " + sum + " "; }
}

class Giver extends Sharer{
   public Giver(String ID, int sum) { super(ID, sum); }

   void share(int n) { other.sum += n;  this.sum  -= n; }   // give to other
}

class Taker extends Sharer{
       public Taker(String ID, int sum) {  super(ID, sum);  }

       void share(int n) { other.sum -= n;  this.sum  += n; }  //  take from other
}

public class Test {

    public static void main(String[] args) {
        Sharer a = new Giver("Geoff", 10);
        Sharer b = new Taker("Tina",  7);
        Sharer c = new Taker("Ted",  15);

        a.other = b;   b.other = c;   c.other = a;
        a.share(2);    b.share(4);    c.share(7);

        System.out.println( a.toString() +  b.toString() + c.toString());
    }
}
```

   (a) What is the output when the `Test` class is run?

   (b) Suppose we were re-define the visibility of the three fields of `Sharer` to be `private` so that the `Giver` and `Taker` no longer have access to these fields.

   Rewrite the methods of `Sharer`, `Giver`, and `Taker` to be consistent with this new definition. You will need to add getters and setters.

7. The modifiers `public, private, protected` define visibility access but there is another visibility level (package, or package-private) which doesn't use a modifier at all. This seems strange. Why do you think the Java language designers didn't use the `package` keyword to indicate that a class/method/field has package visiblity?

Solutions

1. Please see the solution code provided.
2. Please see the solution code provided.
3. When a method is invoked in a program, the method name is stated and the argument (if there is one) has a type. If two methods in a class `C` differ only by the return type, then there is no way in general for the compiler (at compile time) or the JVM (at runtime) to know which method should be used. For example, suppose we had two methods defined in class `C`:

   ```
   A  m(){ ... };          //   returns a type A object
   B  m(){ ... };          //   returns a type B object
   ```

   Let `o` be an object of type `C`. Then, what would `o.m()` return ? You cannot say.
4. The output is

   ```
   toString from B:  default string
   toString from B:  default string
   ```

   Why? The variable `a` references a class `B` object, and so `a.toString()` calls the method `toString()` from class `B`.
5. The compilation error occurs in (e). The problem is that the `A` class has no `foo` method with an argument, and `a` is declared to be of class `A`.

   ```
   Output:
   A             from (c)
   3             from (d)
   A             from (f)    <---  easy to miss this one (see comment below)
   B             from (f)
   11            from (g)
   ```

   Every constructor makes an implicit call to its super class constructor `super()`. Thus, the `B(int n)` constructor implicitly calls the constructor `A()`, which prints out `A`.
6. (a) `Geoff 1 Tina 13 Ted  18`

   (b)
   ```
   // in the Sharer class
   Sharer getOther(){ return  this.other; }
   void   setOther(Sharer other){ this.other = other; }
   int    getSum(){  return  this.sum;   }
   void   setSum(int n){  this.sum  =  n;  }

   // in the Giver class
   void share(int n) {
      this.getOther().setSum( this.getOther().getSum() + n );
      this.setSum( this.getSum()  - n );  }
   }

   //  in the Taker class
   void share(int n) {
      this.getOther().setSum( this.getOther().getSum() - n);
      this.setSum( this.getSum() + n );
   }
   ```
7. The word `package` is used for something else, namely at the top of the file to indicate which package a class belongs to!