Practice for Mid-Semester Test

You should be able to do all questions on this practice test, but it is much longer than you will get in the mid-semester test---this test would be given about 85 minutes.

Part A (21 marks)

Each of the following questions requires a short answer or piece of code. For questions with multiple parts, make sure you **answer every part**.

Question A.1 [4 marks]
Suppose we have classes `Triangle`, `Square`, and `Circle`, which is each a **subclass** of the **abstract** class `Shape`.

a. Write down the definition of an array that can contain *any* of these types of objects (or their subclasses) but not other types of objects.

Shape[] shapes = new Shape[10]

b. Suppose you want the array to contain instances of "pointy" shapes only---i.e. it could contain `Square` and `Triangle` objects but not `Circle` objects.
Use an **interface** called `Pointy` to do this---**do not define any new classes**.
   i. Write down the first line of the new class definitions.
   ii. Write down the new definition of the array.

class Triangle extends Shape implements Pointy { ...
class Square extends Shape implements Pointy { ...
Pointy[] pointyShapes = new Pointy[10]

Question A.2 [1 mark]
Which of the following is FALSE: A class may have more than one

a. Instance
b. Constructor
c. Accessor
d. Mutator
e. Superclass
f. Subclass

Superclass

Question A.3 [2 marks]
Write 2 differences between a constructor and a "normal" method, in about 1 line each**.**

A constructor is never called directly by name; it is only called when we use "new"
A constructor cannot return anything
A constructor must have the same name as the class it belongs to

Question A.4 [2 marks]
For each of the following statements, write whether they are true for **Overriding**, true for **Overloading**, true for **Neither**, or true for **Both**. You don't have to explain your answers.

a. It involves multiple methods with the same name.
b. It involves multiple methods with the same name and same number/type of arguments.
c. It involves multiple methods in the same class.
d. The behaviour of methods may be re-defined.

a. Both; b. Overriding; Overloading; Both

Question A.5 [4 marks]
Assume that `Apple` is a subclass of `Fruit`, and consider the following variable declarations.

```
Apple apple1 = new Apple();
Fruit fruit1 = new Fruit();
Fruit fruit2 = new Apple();
Apple apple2;
```

For each of the following questions, just write the letter(s) as your answer.

a. Which of the assignment statement(s) below will cause a problem **when the program is compiled**?

A

b. Which of the assignment statement(s) below will cause a problem **when the program is executed/run**?

C

A. apple2 = fruit1;
B. fruit2 = apple1;
C. apple2 = (Apple) fruit1;
E. apple2 = (Apple) fruit2;
D. fruit2 = (Fruit) apple1;

Question A.6 [3 marks]
What is printed by the following program:

```
public class Parent {

    public void myPrint(name) {
        System.out.println("parent: " + name);
    }
    class Child extends Parent {
        public void myPrint(name) {
            System.out.println("child: " + name);
```

```
        }
    }
    public static void main(String[] args) {
        Parent p = new Parent();
        p.myPrint("Sam");
        p = new Child();
        p.myPrint("Mary");
    }
}
```

```
parent: Sam
child: Mary
```

Question A.7 [3 marks]
What is printed by the following program:

```
public class ExceptionTest {

    public static void NumberTest(int num) throws RangeException {
        if (num<0 || num>9) {
            throw new RangeException("Number out of range!");
        } else {
            System.out.print("Number is valid.");
        }
    }

    public static void main(String[] args) {
        try {
            NumberTest(-3);
            NumberTest(3);
        } catch (RangeException re) {
            System.out.println("Error!");
        }
        System.out.println("Finish!");
    }
}

class RangeException extends Exception {
    String mesg;
    public RangeException(String s) {
        mesg = s;
    }
    public getMesg() {
        return mesg;
    }
}
```

Error!
Finish!

Question A.8 [2 marks]
In the following code, which method(s) below **must** be implemented in class D to make it a concrete class. Just write the method name(s).

```
abstract class A implements B {
    public abstract void one();
    public void two() { };
}

interface B {
    public void three();
}

interface C {
    public void four();
    public method five();
}

class D extends A implements C {
    …
}
```

one; three; four; five

Part B (10 marks)

Question B.1 [9 marks]

Write a class `ReverseArray` with a method called `reverse` that:

a. Takes as input an array of Object;
b. Copies the **non-null** array objects **in reverse order** into a new array---objects are null are not copied;
c. **Returns** the new array.

Add a `main` method to the class, that:

- Creates an array of `Integer` and calls the `reverse` method;
- Loops through the returned array and prints all values in it.

For full marks, your solution must satisfy the following:

- Null values should not be copied;
- The loop for printing the values from the ArrayList should be a **for-each** loop.

```
class ReverseArray {
   Object[]  reverse(Object[] objectsToCopy) {
      Object[] newObjects = new Object[objectsToCopy.length];
      int newPosn = 0;
      for (int i = objectsToCopy.length-1; i>=0; i--) {
         if (objectsToCopy[i] != null)   newObjects[newPosn] = objectsToCopy[i];
         newPosn++;
      }
      return newObjects;
   }

   public static void main(String[] args) {
      Integer[] ints = {1, 2, 3, null, 4};
      Integer[] newInts = reverse(ints);

      for (Integer i : newInts)
         System.out.print(i);
   }
}
```

Part C: (40 marks)

Your job is to write a system to manage a movie rental system for a movie rental store. Your task here is to maintain the status of movies only, not the customers or other objects.

The movie rental store rents both DVDs and VHS movies. VHS movies are available only for weekly rental. Some DVDs are available for weekly rental; others are only available for daily rental.

The rental fee for a VHS movie is $3 per week. The fee for a weekly DVD is $5 per week. The fee for a daily DVD is $6 per day.

Note: do not try to handle multiple copies of the same film in any special way: if the store has two copies of a movie, there will just be two objects in the system with the same title (but different IDs and rental status).

Question C.1 [13 marks]
Write an **abstract** class *Movie*, including all variables and appropriate methods.

- Every movie stores an *ID* (an Integer) and a *title* (a String); **every movie object must have values for these, and these values cannot be changed**. The ID is unique, and is input to the system (not automatically generated)---it matches a code on the box.
- You can also specify a *year* (an int) for a movie, but this is **optional**.
- The variable *available* (a boolean) has value *true* if the movie is in the store, *false* if it is out on rent.
- The variable *numDaysOut* is an int whose value is the number of days a rented movie has been out on rent.
- For any *Movie*, there will be an accessor *getRentalPeriod()* to return the number of days it can be rented for (i.e. 7 or 1). This method is to be implemented in subclasses since it depends on whether it is a weekly or daily rental.
- Finally, each Movie needs a method *calcLateFee()* to calculate the fee for overdue movies. For simplicity, the late fee of *any* movie (weekly or daily) is calculated by multiplying the number of days it is late by the rental fee. The late fee is not pro-rata: a *weekly* video is charged the full weekly fee for each *day* it is late. For example, a weekly DVD that has been out 9 days will have a late fee of 2 (days late) * $5 = $10. (This method should return 0 if the movie is not overdue). Write this method.
- As well as the standard Movie properties, a DVD stores a boolean *specialFeatures* that tells whether there are special features (e.g. interviews) on the DVD. This **must** be set, for any DVD.

Remember:
- You are to make *Movie* an **abstract** class.
- You will need accessors for variables *ID, title, year, available, numDaysOut, rentalFee, rentalPeriod.*
- Some, **but not all**, of these will also need mutators.
- You will need a method for *calcLateFee().* You are to write this method.
- **Some of these may actually be implemented in sub-classes**.

```
public abstract class Movie {

    private final int ID;   // ID and title should not change once they are set, so are ‘final’
    private final String title;
    private final int year = 0;  // default value since it may not be set
    private boolean available;
    private int numDaysOut = 0;

    public Movie(int ID, String title) {
        this.ID = ID;
        this.title = title;
        available = true;  // when we create it, movie is available
    }
    public Movie(int ID, String title, int year) {    // second constructor handles optional input
        this(ID, title);
        this.year = year;
    }

    // getID(), getTitle(), etc

    public abstract int getRentalPeriod();
    public abstract float getRentalPrice();

    public float calcLateFee() {
        if (numDaysOut > getRentalPeriod() {
           return getRentalPrice() * (numDaysOut – getRentalPeriod());
        } else return 0;
    }

    // it doesn’t say to write the following but they illustrate setting available var
    public boolean rent() {    // return ‘true’ if rent operation was successful
        if (available) {
            available = false;
            return true;
       } else return false;    // operation unsuccessful if movie is not available

    public void return() {
        available = true;
    }

}
```

Question C.2 [15 marks]
Write the subclasses *DVD*, *WeeklyDVD*, *DailyDVD*, *VHS*, appropriately arranged in a hierarchy.

Remember:

- You will need an accessor for *specialFeatures*.
- Some variables listed earlier should be created and/or have values set in these classes.
- Some methods listed earlier may be implemented in these classes.

```
public abstract class DVD extends Movie {
   private boolean specialFeatures;
   //  do NOT repeat the variables stored in superclass!

   public DVD(int ID, String title, boolean specialFeatures) {
      super(ID,title);
      this.specialFeatures = specialFeatures;
   }
   public DVD(int ID, String title, int year, boolean specialFeatures) {
      super(ID,title,year);
      this.specialFeatures = specialFeatures;
   }
}

public class WeeklyDVD extends DVD {

   public WeeklyDVD(int ID, String title, boolean specialFeatures) {
      super(ID,title, specialFeatures);
   }
   public WeeklyDVD(int ID, String title, int year, boolean specialFeatures) {
      super(ID,title,year, specialFeatures);
   }

   public int getRentalPeriod() {
      return 7;
   }
   public float getRentalPrice() {
      return 5;
   }
}

// similarly for DailyDVD and VHS classes
```

Question C.3 [12 marks]

Write a simple *Test* class, containing a method called *test()* which does the following:

1. Declares an array **or** Java Collections Framework object **of your choice** to store movies.
2. Creates the following movies with the following properties (use the methods you have designed above) and adds them to the array/collection:
    a. VHS, title = "On The Waterfront", year = 1954, available = false, days-out = 4
    b. DVD, period = 7 (weekly), title = "Hoop Dreams", available = false, days-out = 10, special-features = true
3. Calls a method *caculateTotalLateFees()*. This method---**which you must write**---loops through the array/collection of all movies and calculates the **TOTAL** late fees owed on all movies stored in the system.

```
public class Test {

   public void test() {

      ArrayList<Movie> movies = new ArrayList<Movie>;
      VHS m1 = new VHS(1, "On the Waterfront", 1954);
      m1.setAvailable(false);   // these set-methods should have been defined above …
      m1.setDaysOut(4);
      DVD m2 = new WeeklyDVD(2, "Hoop Dreams);
      m2.setAvailable(false);
      m2.setDaysOut(10);
      movies.add(m1); movies.add(m2);

      float total = calculateTotalLateFees(movies);
   }

   float calculateTotalLateFees(ArrayList<Movie> movies) {
      float total = 0;
      for (Movie m : movies)  {
         total = m.calculateLateFee();
      }
      return total;
   }
}
```

--- End of Exam Questions ---