Part A (27 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 [3 marks]
There are **three** Java errors in the following simple class definition, on the lines indicated. What are they (no more than one line each). You do not have to correct them; just state in a few words what is wrong.

```
public abstract ClassA {

    final int count = 0;

    public ClassA() {
    }

    public static int putInt(int x) {
        ArrayList integerList = new ArrayList(1);
        integerList.add(0, x);
        x = integerList.get(0);                 // ERROR
        count++;                                // ERROR
    }

    public static void main(String[] args) {
        ClassA a = new ClassA();                // ERROR
    }
}
```

Question A.2 [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.

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.

Question A.3 [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

Question A.4 [2 marks]
List 2 differences between a constructor and a "normal" method, in about 1 line each**.**

Question A.5 [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.

Question A.6 [3 marks]
Some of the Java Collections Framework classes we discussed this semester were ArrayList, HashMap, Vector, and Set. Which type of Collection object would be **best suited** to use for implementing the following. You do **not** have to show an implementation or definition; just name the best JCF type. If you think more than one is equally good, then just choose one of them.

a. A collection of the best-ranked tennis players, in order of ranking, i.e. from highest-ranked to lowest-ranked. The ranking can change weekly and the main task will be to print the players **in rank-order** often.
b. A collection of student enrolment records. The main task will be to **look up student information by unique student id**.

Question A.7 [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**?
b. Which of the assignment statement(s) below will cause a problem **when the program is executed/run**?

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

Question A.8 [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");
    }
}
```

Question A.9 [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;
    }
}
```

Question A.10 [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 {
    …
}
```
.

Part B (15 marks)

Question B.1 [15 marks]

Write a **Generic** class `ReverseArray` with a **Generic non-static** method called `reverse` that:

a. Takes as input an array of objects (of the Generic type);
b. Copies the **non-null** array objects **in reverse order** into an ArrayList (of the Generic type);
c. **Returns** the ArrayList.

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

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

For full marks, your solution must satisfy the following:

- The class and method should each be a **Generic**. If you don't know how to write a Generic class/method, then write the method as best you can without using Generic;
- Null values should not be copied;
- The call within the `main` method must be correct;
- The loop for printing the values from the ArrayList should be a **for-each** loop **or** use an iterator.

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**.

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.

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.

Part D (28 marks)

Specification:
A bank holds many accounts. An account has an account identifier (in the form of an String) and the balance for that account. When an account is created, the bank operator enters, via a graphical user interface, the new account identifier and the opening balance for that account.

Carefully read the following specification, which refers to the GUI in Figure 1. The class diagram in Figure 2 illustrates the design you should follow.

- Figure 1 shows the user interface design for this application;
- When the user clicks the "Add" button, a new instance of an Account is created and added to the allAccounts variable of the instance of Bank.
- When the user clicks the "Save All" button, all accounts that have been created are to be saved to a text file named "allAccounts.txt".

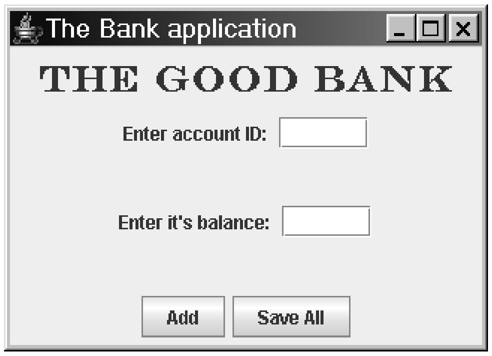


Figure 1: a BankGUI instance when created

Figure 2 below highlights the main classes, variables and methods. Accessor methods are not shown: **you may assume the obvious accessor methods for all private variables**.
(If you need a reminder, the Appendix contains a basic description of a class diagram.)

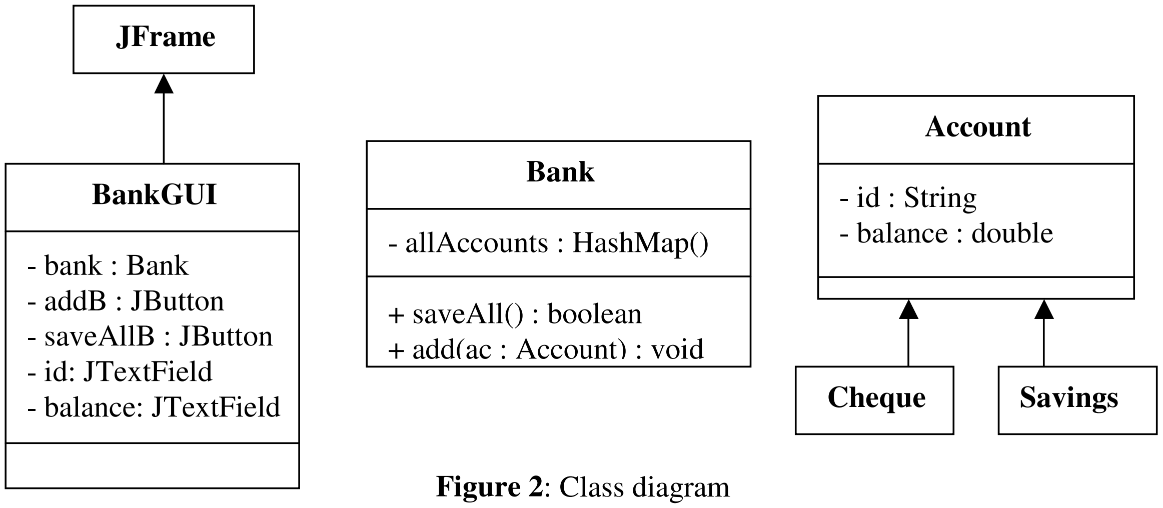


Figure 2: Class diagram

Other information:

- Assume `Account` class has a 2-argument constructor that takes **id** and **balance** as arguments.
- The type of `allAccounts` is as follows, where String is the id: HashMap<String, Account>

Question D.1 [6 marks]
Identify the Java layout managers and components that you would need to use to create the GUI (Graphical User Interface) shown in Figure 1.
Sketch the GUI into your answer booklet and **clearly** indicate on your sketch which parts relate to which layout managers / components.

Question D.2 [8 marks]
Write the code for the saveAll( ) method of the Bank class. This method saves all the Account objects in the `allAccounts` variable into a **text** file called "allAccounts.txt". The format of **each line** of the file should be:

```
      id: 12345; balance: $1000
```

(Of course, the actual id and balance depends on what was stored in that account.) You do not have to worry about formatting the line or lining up columns. It is suggested you use a PrintWriter class.

Question D.3 [13 marks]
To make the GUI work, the BankGUI must implement ActionListener, and each of the buttons (`addB` and `saveAllB`) will have the ActionListener attached to them.
To handle button-click actions, as an ActionListener, the BankGUI will implement the `actionPerformed()` method: this handles all GUI-action events. Its outline is below.

```
      public void actionPerformed(ActionEvent e) {
            String cmd = e.getActionCommand();
            if ( ... ) {
                  ...
            }
      }
```

Complete the actionPerformed() method for BankGUI, handling the button actions.

- You only need to handle 2 actions: clicking the **Add** button or the **Save All** button.
- Assume the `actionCommand` for each button is the same as its text: i.e. "Add" / "Save All".
- You will need to refer to the saveAll() method of the previous question. If you could not write it, just assume it works as it's supposed to.
- You will need to write the snippet of code that handles the action required by Add, including fetching the id/balance info from the text-fields.
- Do not perform any error-checking or validation at all: assume a hypothetical situation where an operator never makes mistakes.

Question D.4 [1 mark]
How is the method `actionPerformed()` called---i.e. who/what calls it?

--- End of Exam Questions ---

Appendix

Selected Standard API

`ArrayList<E>`
- `boolean add(E o)`
- `E get(int index)`
- `boolean isEmpty()`
- `Iterator iterator()`
- `E remove(int index)`
- `int size()`

`HashMap<K,V>`
- `boolean containsKey(Object o)`
- `boolean containsValue(Object o)`
- `V get(Object o)`
- `boolean isEmpty()`
- `Set<K> keySet()`
- `V put(K key, V value)`
- `int size()`
- `Collection<V> values()`

`Iterator<E>`
- `boolean hasNext()`
- `E next()`

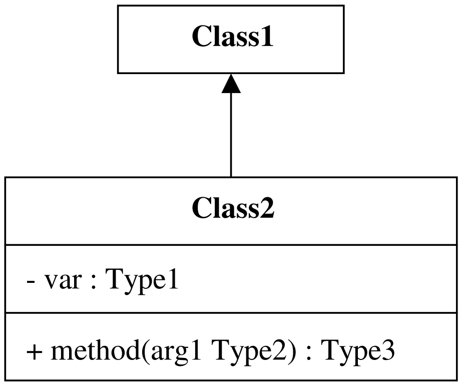
`PrintWriter`
- `PrintWriter(String filename)  throws FileNotFoundException`
- `void close()`
- `void println(String s)`
- `void println(Object o)`

`JButton`
- `String getActionCommand()`
- `void setActionCommand(String actionCommand)`
- `String getText()`
- `String setText()`

`JTextField`
- `String getText()  throws NullPointerException`
- `void setText(String text)`

for-each loop syntax reminder: *`for (Type Item : Collection)`*

Class Diagram Explanation



Explanation of the above diagram:

- `Class1` and `Class2` are names of classes
- The arrow from `Class2` to `Class1` means that **`Class2` is a subclass of (extends) `Class1`**
- `Class2` has variables (in the middle block) and methods (in the bottom block) not in `Class1`
- `var` is the name of a variable of type `Type1`; the – sign means that var is **private** to `Class2`
- `method` is the name of a method, that takes an argument `arg1` of type `Type`. `method` returns values of type `Type3`. The + sign means that `method` is **public**.