RMIT University

Part I Multiple Choice / Fill in the blanks 30 marks

Select the most appropriate response and circle the corresponding letter (A/B/C/D/E). There is only one correct answer for each question.

(2 marks each)

1. **B**
2. **B**
3. **A**
4. **C**
5. **A**
6. **C**
7. **B**
8. **A**
9. **B**
10. **D**
11. **D**
12. **A**
13. **C**
14. **D**
15. **B**

Part II (Program segments) 30 Marks

Write your answers in the space provided

Question 16(3 Marks)

(a) Write a code fragment using a for loop to print the first 20 numbers of the following series on one line.

```
1  2  4  8  16  32  64 128 ...
```

(ie. each number in the series is double the previous number in the series)

```
for (int i = 1; i <= 20; i++)
   System.out.print(i * i + " ");
```

```
1 marks for correct number of repetitions in loop
0.5 mark for correct starting value
0.5 marks for correct use of System.out.print()
0.5 mark for calculating term value correctly
0.5 marks for printing space
```

Question 17(4 Marks)

You are required to complete a code fragment to determine what type of weather warning should be issued for the next day based on projected wind speeds.

A table showing the projected maximum wind speeds and corresponding warnings is shown below.

Warning	**Projected Maximum Wind Speed**
Light Wind Conditions	Up to 15kmh
Moderate Wind Conditions	16kmh to 40kmh
Heavy Wind Conditions	41kmh to 80kmh
Extreme Gale Conditions	Above 80kmh

Write your answer on the next page, by completing the code presented.

```
Scanner  reader = new Scanner(System.in);

System.out.println("Enter the projected wind speed in kmh: ");

double maximumWindSpeed  = reader.nextInt();
String warningMessage;

// now determine the appropriate warning message

if (maximumWindSpeed <= 15)
```

```
   warningMessage = “Light Wind Conditions”;

else if (maximumWindSpeed <= 40)

   warningMessage = “Moderate Wind Conditions”;

else if (maximumWindSpeed <= 40)

   warningMessage = “Heavy Wind Conditions”;
else
   warningMessage = “Extreme Gale Conditions”;

System.out.println(“Wind warning: ‟ + warningMessage);
```

```
   3.0 marks for identifying the categories correctly (1.0 each)
   1.0 marks for assigning the warning messages correctly (0.25
   each)
```

Question 18(5 Marks)

Write a code fragment using nested for-loops to display the pattern below:

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

```
 int numberOfLines = 5;
   int lastValue = 1;

   for (int i = 1; i <= numberOfLines ; i++) // 1 mark
   {
      for (int j = 1; j <= lastValue; j++)// 1 mark
      {
         System.out.print("*");// 1 mark
      }

      lastValue++;                 // 1 mark

      System.out.println();    // 1 mark
   }
// Note that the use of lastValue and numberOfLines are unnecessary.
// Hence give 0.5 for the outer loop for each of the initialisation
// and the condition. Give two marks for the use of lastValue or the
// use of i instead of lastValue in the j-loop condition
```

Question 19 (5 marks)

Complete the missing part of the program below to find the sum of elements in the array numbers.

```
public class TestArray {
   public static void main (String[] args) {

      // array to sum

      int numbers[] = {20,  15,   40,  70,  35 };

      int sum=0; // 1 mark

      for (int i=0; i<5; i++)    // 2 mark
           sum += numbers[i];    // 2 mark


      System.out.println("Sum value of array element is : = "+ sum);
   }
}
```

Question 20 (4 + 3 + 3 + 3 = 13 Marks)

Consider the following partially completed class definition for Balloon and answer the questions that follow.

```
class Balloon {
 private double radius;
 private double volume;
 final double PI = Math.PI;

 public Balloon(){
    volume=0;
   }
...

}
```

(a) Use the two instance variables given to complete the definition of **`addAir(double amount)`** method to add air to the balloon by adding the amount of air to the volume of the balloon and then calculate the radius using the formula r= sqrt (volume/ 4*PI)

```
public void addAir(double amount)
  {
   volume += amount;     // 2 mark

   radius = Math.sqrt(volume / (4 * PI));     // 2 mark
}
```

(b) Write accessors for radius and volume

```
 public double getRadius(){
    return radius;                              // 1.5 mark
   }

  public double getVolume(){
  return volume;                                // 1.5 mark
  }
```

(c) Write a method that print the radius and volume of the balloon with the following header

```
public void print(){
   System.out.println(“radius = “ +  radius());// 1.5 mark
   System.out.println(“volume = “ + volume());// 1.5 mark

 }
```

(d) Given the partial class definition of BalloonTester to test the Balloon class above, answer the questions that follow.

```
public class BalloonTester {

      public static void main(String[] args)
        {
           ...

         }
```

(i) Create an object of Balloon, call it **balloon**

```
Balloon balloon = new Balloon();     // 1 mark
```

(ii) Add air to the object **balloon** of amount 1300

```
balloon.addAir(1300);              // 1 mark
```

(iii) Print out the radius and volume of the object **balloon**

```
balloon.print()                     // 1 mark
```

Part III - Program writing (40 Marks)

Write your answers in the space provided.

Question 21 (40 marks)

A simple application for a LoanBook class and its subclass STLoanBook.

Overview

You are required to provide answers to questions around the theme of a partially completed class, **LoanBook** and its subclass **STLoanBook**, and the system class, **Library**. These questions appear respectively in Sections A, B and C, in the following pages.

The main intention of this section is to allow you to demonstrate your understanding of some or all of the concepts around classes, encapsulation, inheritance and polymorphism.

You are encouraged to add comments to explain your answers where you believe an explanation is necessary. Where the specification is subject to interpretation, you may make any reasonable assumptions but you are required to justify these using comments.

Question 21: The class LoanBook (3 + 3 + 2+ 4 = 12 Marks)

Complete the definition of the class **LoanBook** as per questions that follow and in the spaces provided for your answers.

```
classLoanBook
{
```

(a) In the spaces provided below define instance variables for **ID, title** and **status.**

```
        private String bookID;  //book ID for a loan book 1 mark
        private String bookTitle; // title of a book 1 mark
        private char Status;// Status  for a loan book 1 mark
```

(b) Define a constructor that accepts a book ID and title and sets the appropriate instance variables of the **LoanBook** class. Note that **status** must be set to **'A'** (which means the book is available at construction time). The **borrowerID** instance variable is not set until a book is borrowed.

```
      public LoanBook(String ID, String title) 1 marks
        {
              Status = 'A';
              bookID=ID;
              bookTitle=title;                2 marks

    }
```

(c) Define getters (accessors) for each of the instance variables. The getters should be appropriately named as: `getID() and getStatus()`. **[NOTE TO MARKER: Some students may list getters for all instance variables, only two are expected.]**

```
 public String getID()
{
     return bookID;                 1 marks
}

public char getStatus()             1 marks
{
     return status;
}
```

```
// Given mutators and print() helper methods. The following methods
// are provided.
//
// The borrow method sets the status ‘O’ if the book is available and
// otherwise ignores the request. If the book is successfully
// borrowed then the method also records the ID of the borrower in
// borrowerID

  public void borrow(String borrowerID)
  {
        if (status==‘A’)
        {
              status=‘O’;
              this.borrowerID = borrowerID;

        }
        // no else part because we ignore the request
        // if the book is already on loan
  }


// The setStatus method allows for the status to be altered to
// whatever is passed in the parameter Nstatus

  public void setStatus(char Nstatus)
  {
        status=Nstatus;
  }


// The print() helper method prints only the book properties
// There is no need to print the borrowerID

  public void print()
  {
        System.out.print(bookID + “, “ + bookTitle, “, “, + status);
  }
```

(d) Write a method with the following header for managing the returning of a book. You may assume that the book is on loan.

The method should first change the status of the book to ‘A’. The fine is calculated based on the number of days a book is borrowed. The method should return zero (0) as a fine if a book is borrowed for 7 days or less, and a flat fine of $5 if the book is borrowed for more than 7 days.

```
public int retn(noOfdaysBorrowed)
{
    SetStatus(‘A’);      1 marks

    if(noOfdaysBorrowed>7){     1 marks
         return 5;              1 marks
      }
          else{
           return 0;            1 marks
          }
}
```

Question 22: The LoanBook subclass STLoanBook (1 + 1 + 3+ 3 + 2 + = 10 marks)

In this section all of the questions refer to the subclass of **LoanBook**, named **STLoanBook**. The skeleton code for the STLoanBook class appears below. Answer the questions that follow.

(a) Complete the code below in the space provided to specify that **STLoanBook** is a subclass of **LoanBook**.

```
Class STLoanBook extends LoanBook _// Answer part (a) here   //1 marks
```

(b) Define a new instance variable called **reserverID** of type **String** to keep track of the person, if any, who has placed a reservation on the **STLoanBook** instance.

```
private String reserverID;    //1 mark
```

(c) Write a constructor for **STLoanBook**which takes arguments named **ID**, **title** and appropriately initialises the corresponding instance variables of the newly created object. The **reserverID** instance variable should be set initially to "**NOT RESERVED**". These parameters must be used to initialise the appropriate superclass instance variables through appropriate use of the **super** keyword.

```
public STLoanBook(String ID, String title)    //1 mark
{
   super(ID, title); //1 mark
   reserverID = "NOT RESERVED";    // 1 mark
}
```

(d) Override the **borrow(String borrowerID)** method in this **STLoanBook** subclass. The method should first check to see if a **reserverID** has been recorded for the **STLoanBook** (that is, **reserverID** is not equal to "**NOT RESERVED**") and, if so, ignores the request. Otherwise the method should proceed to update **reserverID** to the given **borrowerID** and invoke the superclass version of the **borrow()** method to complete the loan transaction.

```
// MERCY THIS NEEDS TO CHANGE
public void borrow(String borrowerID )
{
       if( !reserverID.equals("NOT RESERVED"))  { 1 marks
              reserverID = borrowerID;          1 marks
              super.borrow();        1 marks
       }

 }
```

(e) Override the **print()** method of the **LoanBook** given to print all instance variables including additional instance variable of the **STLoanBook** class. (Use the sample output on page 23 as an aid to complete this method.)

```
public void print()
{
  super.print();                            1 marks
  System.out.println(“, “ + reserverID); 1 marks
}

[NOTE TO MARKER: Deduct 0.5 if they do not print the “, “]
```

Question 23: The Library class (5 + 6 + 7 = 18 Marks)

A system class called **Library** class has been partially written to manage a list of up to 50 books. At the end of the code listing you will find a sample output of the program, which shows before and after listings the impact of changes to loan books, to which references are held in the array **books**. Study the application code (class **Library**) and then answer the following questions.

(a) Write the code below to add the new book objects shown in the AFTER listing. These are the objects:

```
B129, Soil Mechanics    // LoanBook object
B342, Cost Accounting,  // STLoanBook object
```

```
books[bookCount]= new LoanBook("B129", "Soil Mechanics"); //1.5 marks
bookCount++; // 1 mark

books[bookCount]= new STLoanBook("B342", "Cost Accounting"); //1.5 marks
bookCount++;// 1 marks
```

(b) Write a fragment of code to list all books that are **STLoanBook** objects.

```
for (int i = 0; i < bookCount; i++)   //2 marks
         books[i]). print();          // 3 marks
         System.out.println();       // 1 mark
     }
```

(c) Complete the following code which should display all details of the book from the array, if found. If the book matching the ID that is input, the report should also print what kind of book (LoanBook or STLoanBook) it is. If no such booking is found, the report should display "No book with given ID".

```
System.out.print("Enter book ID: ");
ID = reader.nextLine(); // reader is a Scanner object
// provide the remaining code
```

```
  [NOTE TO MARKER: As there are several possible solutions to this type
   of question, distribute the marks into two categories: 3.5 for
   searching the array and 3.5 for reporting.

  LoanBook book = null; // 0.5 marks
 for (int i = 0; i < bookCount; i++)  //1 mark
 {
   if (ID.equals(books[i].getID()))    //1 marks
   {
      book = books[i]; // 1 mark
   }
 }

 if(book==null)                 // 1 mark
       System.out.println("Error – No book with ID")
 else
 {
      book.print(); // 0.5 mark
      if(book instanceof STLoanBook) // 1 mark
           System.out.println("STLoanBook");  //0.5 mark
      else
           System.out.println("LoanBook"); // 0.5 mark
  }
```