BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
FIRST SEMESTER 2018-2019
MID SEMESTER EXAM (CLOSED BOOK TEST)
OBJECT ORIENTED PROGRAMMING (CS F213)

DATE: 12th October 2018 **MAX MARKS: 60** **WEIGHTAGE: 30%** **TIME: 90 Min**

Important Instructions:

1. Attempt each question on a fresh page
2. Assume that all the necessary packages and classes are already imported in your program.
3. Assume that all the necessary getter and setters are already provided.

Q1. The Library class has **String bookName, int bookID, String rackNo** as its data members. Our application is used for searching the rack number of the book either by the book name or by the book ID. The search is done using a recursive binary search. The **split()** method is used to retrieve an array list of book IDs or book names from the Library objects sorted in descending order. This array list is given as an input to the **binarySearch()** method.

The recursive **binarySearch()** is described below. It assumes that the elements are sorted in ascending order.

1. Let low and high denote the left and the right index of the search space. Initially let low = 0 and high = number of objects.
2. Find the middle element and compare it with the key (element you are searching for).
3. If the key matches with the middle element, return the index of the middle element.
4. Else, if the key is greater than the middle element, then the key can only be found in the right half of the search space. Make a recursive call to the binarySearch() method on the right half of the search space after the middle element.
5. Else, if the key is smaller than the middle element, the recursive call is made on the left half of the search space before the mid element.
6. When the left index is greater than the right index, print a message "Book not found".

Let the **Library** class implements a Search interface as shown below:

```
interface Search {
public static <T extends Comparable<T>> int binarySearch(List<T> list, T key, int low,
int high)     {
```

/* Q1. (a) Write your code to perform a generic binary search algorithm using recursive approach. Let the arguments to the generic method be: a generic list of type T, key of type T, low and high indexes of the search space. Let T be bounded by the Comparable interface. */   }   **[05 Marks]**

```
static ArrayList split (List<Library> list, int code)     {
```

/* Q1. (b) Let the split method takes a list of type Library and an integer code as argument. This method is used to retrieve all the book IDs or the book names from the input list to form an integer array list of book IDs if the code == 0 or a string array list of book names when the code == 1. */   }   **[05 Marks]**

```
}

class Library implements Search{
      String bookName;
      int bookID;
      String rackNo;
      Library(String name, int id,String num){
            bookName = name;
            bookID = id;
            rackNo = num;     }
      public String toString() {
         return "Name: "+bookName+" ID: "+bookID+" Rack No.: "+rackNo;  }
}
class TestLibrary{
public static void main(String args[]){
      int index;
      ArrayList<Library> al = new ArrayList<Library>();
```

/* Q1. (c) Create four Library objects and add them to the ArrayList */ **[02 Marks]**

/* Q1. (d) Sort the Library objects in descending order by bookName using an anonymous instance of the Comparator */ **[04 Marks]**

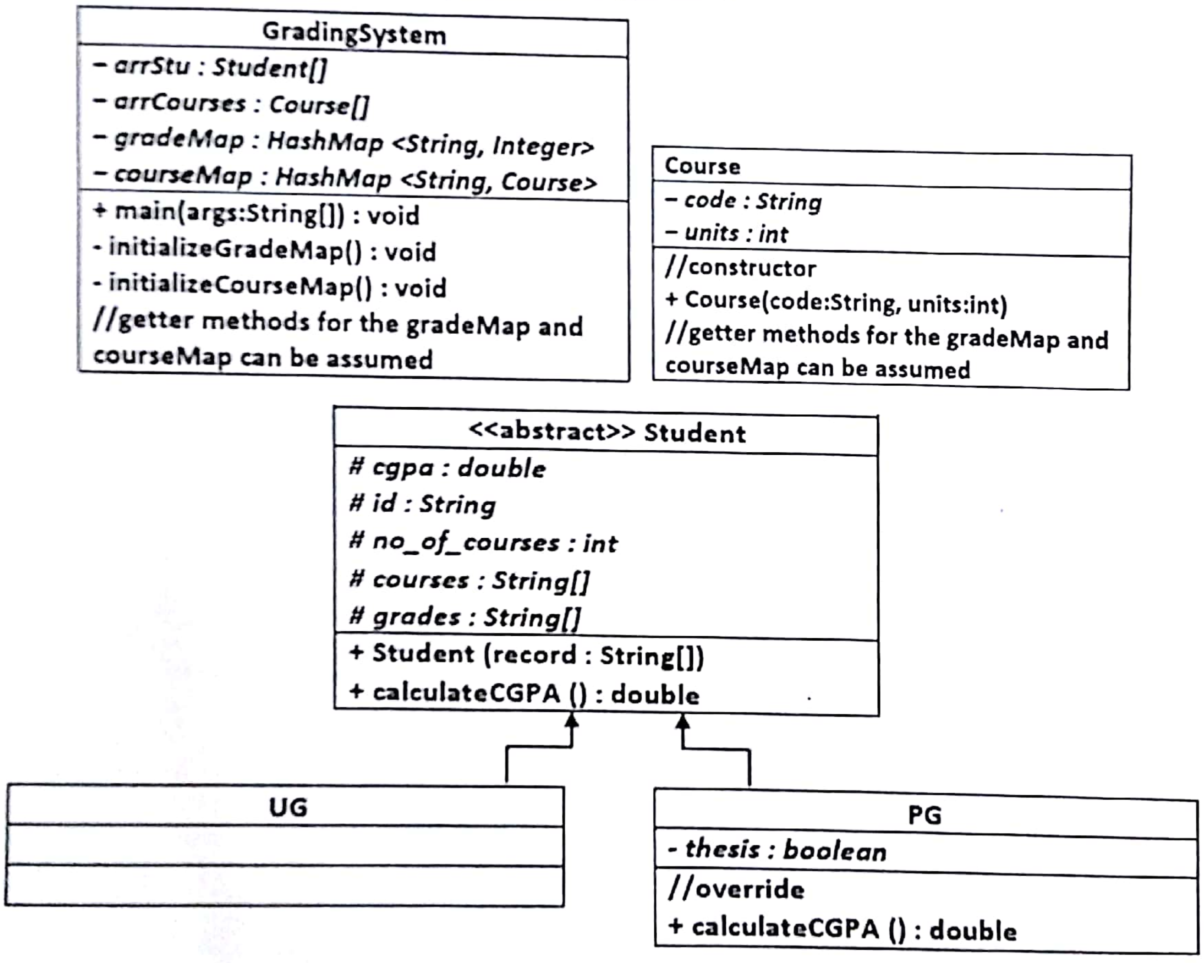
```
ArrayList<String> S = new ArrayList<String>();
S = Search.split(al,1);
index = Search.binarySearch(S, "C++", 0, al.size()-1);
if (index!=-1)
        System.out.println(al.get(index));
```

/* Q1. (e) Sort the Library objects in descending order by bookID using an anonymous instance of the Comparator */ **[04 Marks]**

```
ArrayList<Integer> I = new ArrayList<Integer>();
I = Search.split(al,0);
index = Search.binarySearch(I, 347888, 0, al.size()-1);
if (index!=-1)
        System.out.println(al.get(index));   }

}
```

Q2. The UML diagram of the student grading system is shown below in Fig. 1. Followed by the diagram the description of each class is provided. Carefully read the specification of each class.



GradingSystem
– arrStu : Student[]
– arrCourses : Course[]
– gradeMap : HashMap <String, Integer>
– courseMap : HashMap <String, Course>
+ main(args:String[]) : void
- initializeGradeMap() : void
- initializeCourseMap() : void
//getter methods for the gradeMap and courseMap can be assumed
Course
– code : String
– units : int
//constructor
+ Course(code:String, units:int)
//getter methods for the gradeMap and courseMap can be assumed
<<abstract>> Student
# cgpa : double
# id : String
# no_of_courses : int
# courses : String[]
# grades : String[]
+ Student (record : String[])
+ calculateCGPA () : double
UG
PG
- thesis : boolean
//override
+ calculateCGPA () : double

Fig. 1: UML diagram of University Grading System

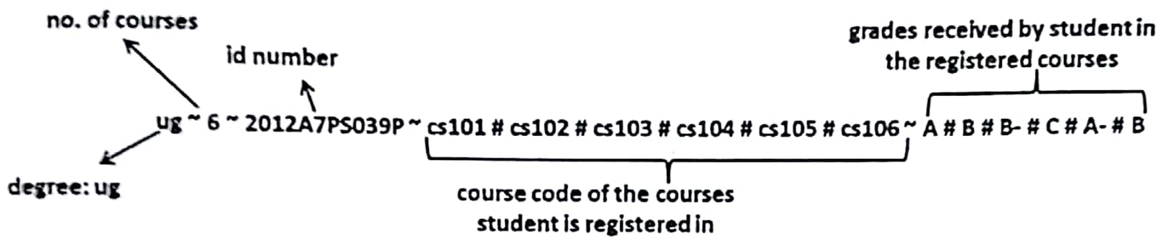
(I) class GradingSystem: All the attributes and methods of this class are static. The attributes are shown in the UML diagram and the specification of the methods is given below:

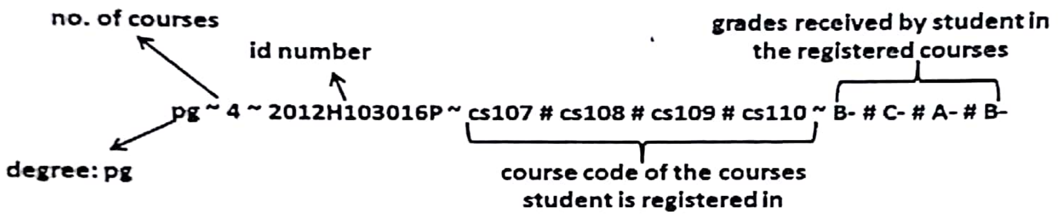
Methods:

(a) main() – this method reads the two "`.txt`" files viz. "`courses.txt`" and "`student.txt`" to initialize and populate the two arrays named `arrCourses:Course[]` and `arrStu:Student[]`. After that the main method makes a call to `initializeGradeMap()` and `initializeCourseMap()` methods that are explained later. In end the cgpa of all the students is calculated by giving a polymorphic call to the method lculateCGPA():double of the Student class. **The content present in the file "courses.txt" is in the ormat "course code ~ no of units"**.

```
For example
cs101~3
cs102~4
cs103~5
cs104~4
...
Thesis~16
```

The content present in the file "student.txt" is in two different formats that are described below:


no. of courses
id number
grades received by student in the registered courses
ug ~ 6 ~ 2012A7PS039P ~ cs101 # cs102 # cs103 # cs104 # cs105 # cs106 ~ A # B # B- # C # A- # B
degree: ug
course code of the courses student is registered in


no. of courses
id number
grades received by student in the registered courses
pg ~ 4 ~ 2012H103016P ~ cs107 # cs108 # cs109 # cs110 ~ B- # C- # A- # B-
degree: pg
course code of the courses student is registered in

More students' records in the text file "student.txt" are shown below:

```
ug~6~2012A7PS039P~cs101#cs102#cs103#cs104#cs105#cs106~A#B#B-#C#A-#B
ug~5~2012A7PS049P~cs101#cs102#cs103#cs104#cs105~A-#B-#B-#C-#A-
ug~7~2012A7PS059P~cs101#cs102#cs103#cs104#cs105#cs106#cs107~A#A#B-#C#A-#B-#A
...
...more ug records
pg~4~2012H112089P~cs107#cs108#cs109#cs110~A#B#C#A
pg~1~2012H112091P~Thesis~A-
...
...more pg records
```

IMPORTANT: It should be noted that different students (both ug and pg) may be registered in different number of courses. **[Hint: This information is useful while initializing the student object]**.

(b) initializeGradeMap() – this method initializes `gradeMap:HashMap<String, Integer>` attribute. This map contains the mapping between different grades and their multiplication factors. For example,

A→10, A- →9, B →8, B- →7, C →6, C- →5, D →4, E →2 [We are not considering NC grade].

(c) initializeCourseMap() – this method initializes `courseMap:HashMap<String, Course>` attribute after obtaining information from `arrCourses:Course[]` attribute.

(d) Getter and setters for the `gradeMap` and `courseMap` can be assumed.

Q2. (a) Implement the class GradingSystem and all of its methods described above. **[10 Marks]**

(II) class Course: All the attributes and the methods of this class are shown in the UML diagram of Fig.1. **[This class can be assumed to be implemented already for your use].**

(III) class Student: This is an abstract class. All of its attributes and methods are shown in the UML diagram of Fig.1 and are self-explanatory. The constructor of this class receives a student record i.e. `record:String[]` whenever the student object is instantiated and initializes all the fields. Also this class has a method named **`calculateCGPA():double`** which calculates the CGPA of the student. This method uses the `gradeMap:HashMap<String, Integer>` and the `courseMap:HashMap<String, Course>` attributes of the `GradingSystem` to fetch information related to the number of units w.r.t a given grade and the Course corresponding to the course code.

Q2. (b) Implement the class Student and all of its methods described above. [05 Marks]

(III) class PG: This class inherits from **class Student**. But it has one additional attribute `thesis:boolean`. If the attribute is set, it means the student is registered in the course with the course code **'Thesis'**. In that case the class overrides the **calculateCGPA():double** method of the parent class and checks if the multiplication factor corresponding to the **Thesis** course is greater than or equal to 8 (which means the student has received a grade B or better), then it adds additional 0.1 to the CGPA. Otherwise, the parent class implementation is used.

Q2. (c) Implement the class PG and all of its methods described above. [04 Marks]

(IV) class UG: This class inherits from **class Student**. It has no additional method or attribute of its own.

Q2. (d) Implement the class UG. [01 Mark]

Q3. What will be the output of the following code execution? [04 Marks]

```
public class Test {
        int data[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
public static void main(String args[]) {
        Test foo = new Test();
        foo.execute();
}
public void execute() {
        try {
                System.out.println(testIt(5)); ✓
                System.out.println(testIt(11));
                System.out.println(testIt(-3));
        }
        catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Index exception");
        }
        System.out.println("Goodbye World");
}
int testIt(int index) {
        try {
                return data[index];
        }
        catch (ArrayIndexOutOfBoundsException e) {
                if (index < 0) throw e;
                else return 10;
        }
        finally {
                System.out.println("leaving testIt");
        }
    }
}
```

Q4. Will the following code compile? If not correct the error and predict the output. [03 Marks]

```
class Test{
static int y=0;
        void method(int y) {
                class innerClass {
                        public int x=1;
                                void innermethod() {
                                        System.out.println("value of x and y = "
                                                +this.x+", "+this.y+", "+~~HelloWorld.y~~);
                                                                          Test.y
                                }
                }
                innerClass in = new innerClass();
                in.innermethod();
        }
        public static void main( String args[] )
        {
                Test                       Test
                ~~HelloWorld~~ t=new ~~HelloWorld~~();
                t.method(55);
        }
```

Scanned by CamScanner

```
}
```

Q5. Assume that the following code passes the Java typechecker: **[03 Marks]**

```
A a = new A();
B b = new C(a);
D d = b;
```

Mark all of the following that must be true:

(a) A is a supertype of B
(b) A is a subtype of B
(c) A is a supertype of C
(d) A is a subtype of C
(e) A is a supertype of D
(f) A is a subtype of D
[✓ (g) B is a supertype of C]
(h) B is a subtype of C
[✓ (i) B is a supertype of D]
[✓ (j) B is a subtype of D]
(k) C is a supertype of D
[✓ (l) C is a subtype of D]

Q6. Java contains two types of exceptions checked and unchecked. A checked exception either has to be caught or it needs to be thrown. Why all exceptions are not treated this way? Why there are unchecked exceptions? **[03 Marks]**

Q7. Predict the output of the following codes. **[02+03+02 Marks]**

(a)
```
class Parent {
        void say(List<String> list) {
                System.out.println("Parent");}
}
class Child extends Parent {
        void say(LinkedList<Integer> list) {
                System.out.println("Child");}
}
public class Test  {
        public static void main(String args[]) {
                Parent c = new Child();
                c.say(new LinkedList<String>());
        }
}
```

(b)
```
class Test{
        public static void main (String[] args)
          {
            short s1 = 10;
            Short s2 = 20;
            aMethod(s1, s2);
            aMethod(s2);
            aMethod(s2, s2);
          }
          static void aMethod (short x, short y)
          {
            System.out.println("short version: " + x);
          }
          static void aMethod (long x)
          {
            System.out.println("long version: " + x);
          }
          static void aMethod (Short... x)
          {
            System.out.println("Var args version: " + x);
          }
}
```

(c)
```
class Test{
        public static void main(String[] args) {
                StringBuffer s=new StringBuffer("String Buffer.");
                int p=s.length();
                int q=s.capacity();
                System.out.println("Length: "+p);
                System.out.println("Capacity: "+q);
        }
}
```