Letterkenny Institute of Technology

Course code: OOPR CP603

YEAR 2 COMPUTING

(Common paper for all streams)

Subject: Object Oriented Programming **Stage:** 2

Date: January 2018 **Examiners:** Mr. T. Devine
Dr. A. Belatreche

Time allowed: 3 hours

INSTRUCTIONS

Answer any FOUR questions.

NOTE: It may be useful to remove the appendices from this paper for easy reference.

Question 1

Appendix A has partial code that implements a basic grid-based Noughts and Crosses game that was introduced in the module.

(a) In the `Board` class declare and create a 2 dimensional (3x3) class array called `board` to store 9 integers and provide the code in `clearBoard()` that initialises each value in the array to 0 (ZERO).
(8 marks)

(b) Identify any shared class variable(s) and method(s) that both `Human` and `Computer` classes use.
(3 marks)

(c) Write the code for a superclass called `Player` to implement the shared variable(s) and method(s) identified in (b).
(7 marks)

(d) Rewrite the signatures for the `Human` and `Computer` classes to become subclasses of the `Player` class.
(2 marks)

(e) Write an `AdvancedComputer` class which is a subclass of `Computer` and override the `setMove()` method using the overrides annotation. There is no need to provide code inside the overridden method.
(5 marks)

Question 2

Appendix B shows an `Asteroid` class being used by a tester class `AsteroidTester`.

(a) Rewrite the code to throw and catch an `Exception` object if the asteroid created is fully or partially outside a window size of (400,400). Print the exception message when caught.
(6 marks)

(b) Provide the code for a user-defined exception class called `AsteroidOutOfBounds`. It should print an exception message "asteroid out of bounds".
(5 marks)

(c) Rewrite the code to throw and catch the class created in (b).
(6 marks)

(d) Distinguish between the exception keywords `throw` and `throws`.
(4 marks)

(e) Briefly describe the concept of exception propagation.
(4 marks)

Question 3

Appendix C shows a class diagram for the `CelestialObject`, `Planet` and `Moon` classes.

(a) *Shadowing class variables is a common mistake for programmers new to inheritance.* What does this mean? Provide an example from Appendix C.

(4 marks)

(b) Given the statement in (a) explain how you would correct the mistake made in the class diagram in Appendix C.

(2 marks)

(c) Describe the purpose of the `super` keyword.

(2 marks)

(d) The `Planet` class has 2 methods called `addMoon()`:

(i) Name this common object-oriented method concept.

(2 marks)

(ii) Provide the code to implement both methods. Assume a planet uses an array list to store moons.

(7 marks)

(e) Given what you know about the `super` keyword and shadow class variables, write the correct code to implement the constructors for the classes `Planet` and `Moon` in Appendix C.

(8 marks)

Question 4

(a) What is wrong with the `Habitable` interface in Appendix D (a)?

(2 marks)

(b) Rewrite the code given in Appendix D (a) so `Planet` implements `Habitable` correctly.

(4 marks)

(c) Update the `Planet` class again with the appropriate method so the code sample in Appendix D (b) will work properly.

(8 marks)

(d) Examine the use of the `Collections` class method `sort()` in Appendix D (c). Provide the code for the missing class `RadiusComparator` that is used to sort planets in ascending order by radius. Use the `Comparator` interface given in Appendix D (d).

(9 marks)

(e) In general, when would you use the `Comparable` interface instead of the `Comparator` interface.

(2 marks)

Question 5

```
int[] list = {17, 26, 5, 2};
```

(a) Given the array `list` above, how many *passes* are required for a bubble sort algorithm to sort the values in ascending order.

(3 marks)

(b) Using `list` clearly show the state of the array after each *pass* of a bubble sort algorithm used to sort the array values in ascending order.

(6 marks)

(c) Appendix E contains code for a bubble sort. Provide the missing code in the method `swap()`.

(9 marks)

(d) For the `sequentialSearch()` method in Appendix E, provide the missing code that implements a sequential search algorithm.

(7 marks)

Question 6

Describe using code examples each of the following concepts:

(a) Polymorphism
(b) `@Override` annotation
(c) `throws` keyword
(d) `this` keyword
(e) `private` access modifier

(25 marks)

Appendix A

```
//
// Board.java
//
public class Board
{
  // Q1(a)
  // declare a 2d board array
  . . .

  public Board()
  {
    // Q1(a)
    // create 2d board array & clear the board
    . . .
    clearBoard();
  }


  // clear (0) all values in board array
  public void clearBoard()
  {
    // Q1(a)
    . . .
    . . .
    . . .
  }

}
```

Appendix A continued

```
//
// Human.java
//
public class Human // Q1(d)
{
  private int[] move = new int[2];

  public Human()
  {
    System.out.println("Human Player created!");
  }

  public void setMove(int x, int y)
  {
    this.move[0]=x;
    this.move[1]=y;
  }

  public int[] getMove()
  {
    return this.move;
  }
}

//
// Computer.java
//
public class Computer // Q1(d)
{
  private int[] move = new int[2];

  public Computer()
  {
    System.out.println("Computer Player created!");
  }

  public void setMove(int x, int y)
  {
    this.move[0]=x;
    this.move[1]=y;
  }

  public int[] getMove()
  {
    return this.move;
  }
}
```

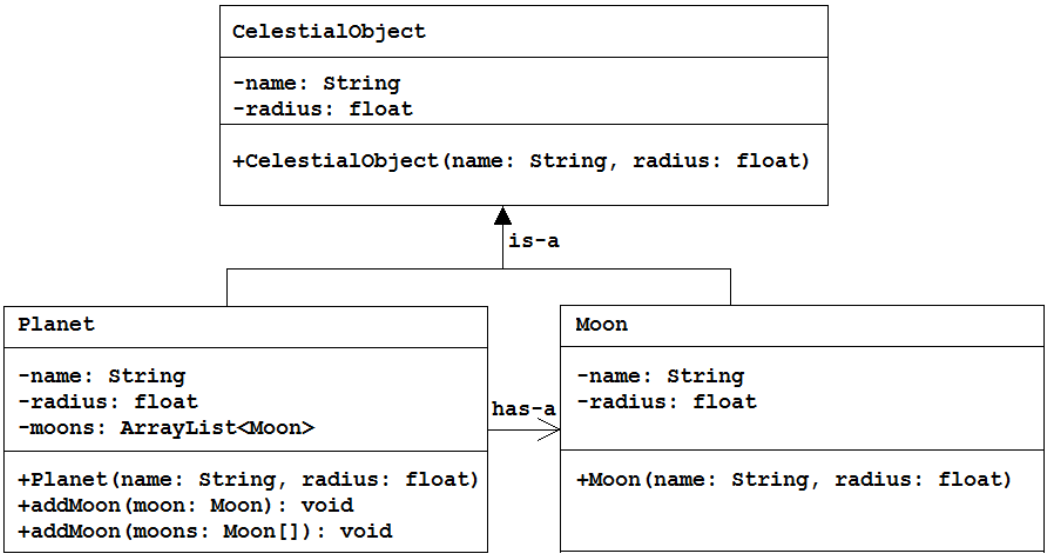
Appendix B

```
public class Asteroid
{
  private int x;
  private int y;
  private int radius;

  public Asteroid(int x, int y, int radius)
  {
    this.x=x;
    this.y=y;
    this.radius=radius;
  }
}

public class AsteroidTester
{
  public static void main(String[] args)
  {
    Asteroid asteroid = new Asteroid(200,200,50);
  }
}
```

Note:

```
if (x+radius>400 || x-radius<0 || y+radius>400 || y-radius<0)
  // outside window
```

Appendix C


CelestialObject
-name: String
-radius: float
+CelestialObject(name: String, radius: float)
is-a
Planet
-name: String
-radius: float
-moons: ArrayList<Moon>
+Planet(name: String, radius: float)
+addMoon(moon: Moon): void
+addMoon(moons: Moon[]): void
has-a
Moon
-name: String
-radius: float
+Moon(name: String, radius: float)

Appendix D

(a)

```
public interface Habitable
{
  boolean isHabitable()
  {
    return true;  // assume it always returns true
  }
}

public class Planet
{
  private String name;
  private float radius;

  public Planet(String name, float radius)
  {
    this.name=name;
    this.radius=radius;
  }
}
```

(b)

```
Earth earth = new Planet(“Earth”,5.0));
Mars mars = new Planet(“Mars”,3.0);

if(earth.equals(mars))
  System.out.println(“equals”);
else
  System.out.println(“!equals”);
```

(c)

```
public class Tester
{
  public static void main(String[] args)
  {
    ArrayList<Planet> planets = new ArrayList<Planet>();
    planets.add(new Planet(“Earth”,5.0));
    planets.add(new Planet(“Mars”,3.0));

    Collections.sort(planets, new RadiusComparator());
  }
}
```

Appendix D continued

(d)

Interface Comparator<T>

Type Parameters:

T - the type of objects that may be compared by this comparator

Method Summary

Methods

Modifier and Type	Method and Description
int	compare(T o1, T o2) Compares its two arguments for order.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this comparator.

Interface Comparable<T>

Type Parameters:

T - the type of objects that this object may be compared to

Method Summary

Methods

Modifier and Type	Method and Description
int	compareTo(T o) Compares this object with the specified object for order.

This is how the Comparable interface would be written:

```
public interface Comparable<T>
{
   public int compareTo(T o);
}
```

Appendix E

Sorting

```
public static void bubbleSort(int[] array)
{
  for(int i= 0;i<array.length-1;i++)
  {

    for(int j=0;j<array.length-i-1;j++)
    {
      if(array[j]<array[j+1])
      {
        // swap the adjacent elements
        swap(array,j+1,j);
      }
    }
  }
}

public static void swap(. . .)
{
  . . .
  . . .
}
```

Searching

```
int[] list = {17, 26, 5, 2}
int findValue=26;

if(sequentialSearch(list, findValue)==-1)
   System.out.println(findValue + " NOT found");
else
   System.out.println(findValue + " found");


// return -1 if not found, return index if found
public static int sequentialSearch(... , ...)
{
  . . .
  . . .
}
```