Letterkenny Institute of Technology

Course code: OOPR CP603

YEAR 2 COMPUTING

(Common paper for all streams)

Subject:	Object Oriented Programming	**Stage:**	2
Date:	January 2017	**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 code that implements a basic grid-based game that was introduced in the module.

(a) In the `World` class `createEnemy()` method provide the code so the enemies appear at a random location rather than the bottom-right corner.

(4 marks)

(b) In the `Player` class you are given the code to allow the player to move north. Provide the code to:

(i) Move the player south

(3 marks)

(ii) Move the player south-west

(3 marks)

(c) Identify a shared class variable and method that both `Player` and `Enemy` classes use. Then:

(i) Write the code for a superclass called `GameCharacter` to implement the shared variable and method.

(4 marks)

(ii) Rewrite the signatures for the `Player` and `Enemy` classes to become subclasses of this new class.

(2 marks)

(d) Write an `AdvancedPlayer` class which is a subclass of `Player` and override the `Player` method `move()` using the overrides annotation. There is no need to provide code inside the overridden method.

(4 marks)

(e) Write a new method for the `World` class called `printEnemies()` which will print the coordinates of each enemy character to the screen. The output should look like this if all enemies are at (5,5):

```
--Enemies--
[X: 5 Y: 5]
[X: 5 Y: 5]
[X: 5 Y: 5]
[X: 5 Y: 5]
```

(5 marks)

Question 2

A `Planet` class is given:

```
public class Planet
{
  private String name;
  private int radius;

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

(a) Override the `toString()` method.

(2 marks)

(b) Modify the `Planet` class so the code snippet below will work:

```
ArrayList<Planet> planets = new ArrayList<Planet>();

planets.add(new Planet("Earth", 150));
planets.add(new Planet("Mars", 50));

if (planets.contains(new Planet("Mars", 50)))
{
  //Do something
}
else
{
  //Do something else
}
```

(8 marks)

(c) Assuming that you want to use the `sort` method of the `Collections` class to sort the `planets` list alphabetically by name:

```
Collections.sort(planets);
```

Provide the required changes to the `Planet` class. See Appendix B.

(6 marks)

(d) If you wanted to provide the additional capability of sorting by radius, provide a mechanism that will allow this. Show how you would invoke the `Collections.sort()` method for this. See Appendix B.

(9 marks)

Question 3

Appendix C shows the `Planet` and `Moon` classes as a Java composition – a planet *has-a* number of moons.

(a) Provide the code for each of the `Moon` class constructors. (6 marks)

(b) Provide the code for the `printMoons()` method. Assume that all other methods are complete. (5 marks)

(c) Give the test code to create a `Planet` object with two moons. The planet should be Mars with a radius of 50. The moons of Mars are:
- Phobos, with a radius of 10.0 and orbital period of 28
- Deimos with a radius of 2.5 and orbital period of 42 (6 marks)

(d) Complete the code in (c) to show how you could search for the moon named Deimos and if found change the orbital period to 15. Your search must get the moon information from the planet object. Assume that all other methods are complete. (8 marks)

Question 4

(a) What is an abstract class? (3 marks)

(b) Modify the code in Appendix D (a) to make the method `display()` abstract. (3 marks)

(c) Using the documentation in Appendix D (b), (c), (d):

(i) Provide the code for the interface `Habitable`. (3 marks)

(ii) Modify the `Planet` class to implement the functionality necessary for the interface `Habitable`. A planet is habitable if it has a `distance` value greater than 200 and less than 500. (5 marks)

(iii) Complete the tester code that tests your habitable planet functionality works. (2 marks)

(d) For the selection sort algorithm used in the module, how many comparisons would be required for a list of seven elements? (3 marks)

(e) For the binary search algorithm shown in Appendix D (d), list the indices of the elements that would be examined when searching for the value 30 in the list:

[2, 5, 7, 13, 15, 16, 18, 20, 22, 24, 30, 34, 39]

Clearly identify the values in variables `mid`, `start`, and `end`. (6 marks)

Question 5

Appendix E contains a class diagram for a `SolarSystem` class. `SolarSystem` uses the `Planet` and `Moon` classes shown in Appendix C. Assume all the code for `Planet` and `Moon` is implemented. Provide the following code for the `SolarSystem` class:

(a) An `ArrayList<Planet>` reference called `planets.`

(2 marks)

(b) An `addPlanet()` method which accepts a planet object and will add it to the arraylist.

(3 marks)

(c) A method called `listPlanets()` which uses an enhanced for loop to print each planets details.

(4 marks)

(d) A `withinRadius()` method which will have the following method signature:

```
public ArrayList<Planet> withRadius(float value)
```

Return an arraylist of planets whose radius is less than value passed to the method. If there are no planets return an empty arraylist.

(8 marks)

(e) A `searchByMoonName()` method which has this signature:

```
public Planet searchByMoonName(String value)
```

It should return a reference to the planet with that moon name. If there is no match return a `null` value.

(8 marks)

Question 6

Describe using code examples each of the following concepts:

(a) Encapsulation (5 marks)
(b) `@Override` annotation (5 marks)
(c) `instanceof` operator (5 marks)
(d) `super` keyword (5 marks)
(e) `protected` access modifier (5 marks)

Appendix A

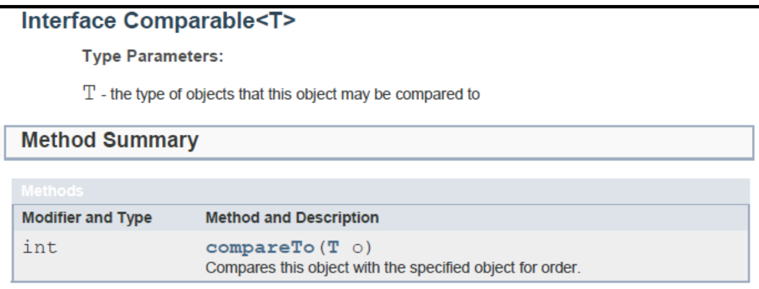
```
//
// World.java
//
public class World
{
  public static final int MAX_CELL=5;
  public static final int MIN_CELL=0;
  public static final int KEY_NORTH=38;
  public static final int KEY_SOUTH=40;
  public static final int KEY_WEST=37;
  public static final int KEY_EAST=39;
  public static final int KEY_NW=36;
  public static final int KEY_NE=33;
  public static final int KEY_SW=35;
  public static final int KEY_SE=34;
  private ArrayList<Enemy> enemies = new ArrayList<Enemy>();
  private Player player;

  public World(Player player, int noOfEnemies)
  {
    this.player = player;
    for (int i = 0; i < noOfEnemies; i++)
      createEnemy(i+1);
  }

  private void createEnemy(int enemyNumber)
  {
    // Q1 - (a)
    enemies.add(new Enemy(enemyNumber, new
    Location(World.MAX_CELL, World.MAX_CELL)));
  }

  public void update(int keycode)
  {
    player.move(keycode);
    for (Enemy e : enemies)
      e.move();
    player.updateScore(1);
    println("Score="+player.getScore());
  }

  public void drawWorld()
  {
    // draws grid
  }

  public Player getPlayer()
  {
    return player;
  }

  // Q1 - (e)
}
```

```
//
// Player.java
//
public class Player // Q1 - (c) (ii)
{
  private Location location;
  private int score;

  public Player(Location location)
  {
    this.location = location;
    this.score=0;
  }

  public Location getLocation()
  {
    return location;
  }

  public int getScore()
  {
    return score;
  }

  public void updateScore(int value)
  {
    this.score+=value;
  }

  public void move(int keycode)
  {
    switch(keycode)
    {
    case World.KEY_NORTH:
      location.changeY(-1);
      if (location.getY()<World.MIN_CELL)
        location.setY(World.MAX_CELL);
      break;

    // Q1 - (b)
    }

  }
}
```

```
//
// Enemy.java
//
public class Enemy // Q1 - (c) (ii)
{
  private int enemyNumber;
  private Location location;

  public Enemy(int enemyNumber, Location location)
  {
    this.enemyNumber = enemyNumber;
    this.location = location;
  }

  public int getEnemyNumber()
  {
    return enemyNumber;
  }

  public Location getLocation()
  {
    return this.location;
  }

  public void move()
  {
    int direction =
    (int)random(World.KEY_WEST,World.KEY_SOUTH);
    switch(direction)
    {
    case World.KEY_NORTH:
      location.changeY(-1);
      if (location.getY()<World.MIN_CELL)
        location.setY(World.MAX_CELL);
      break;
    }
  }
}
```

```
//
// Location.java
//
public class Location
{
  private int x;
  private int y;

  public Location(int x, int y)
  {
    this.x = x;
    this.y = y;
  }
  public int getX()
  {
    return x;
  }
  public int getY()
  {
    return y;
  }
  public void setX(int x)
  {
    this.x = x;
  }
  public void setY(int y)
  {
    this.y = y;
  }
  public void changeX(int amountToChange)
  {
    this.x += amountToChange;
  }
  public void changeY(int amountToChange)
  {
    this.y += amountToChange;
  }
  public String toString()
  {
    return "[X: " + x + " Y: " + y + "]";
  }
}
```

Appendix B



Interface Comparable<T>

Type Parameters:

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

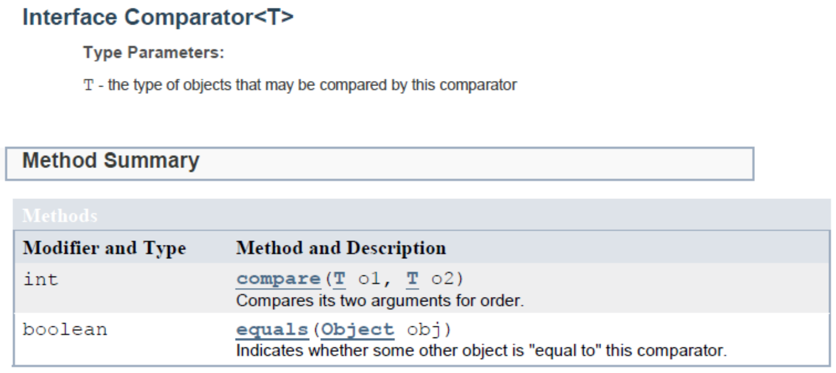
Method Summary

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);
}
```



Interface Comparator<T>

Type Parameters:

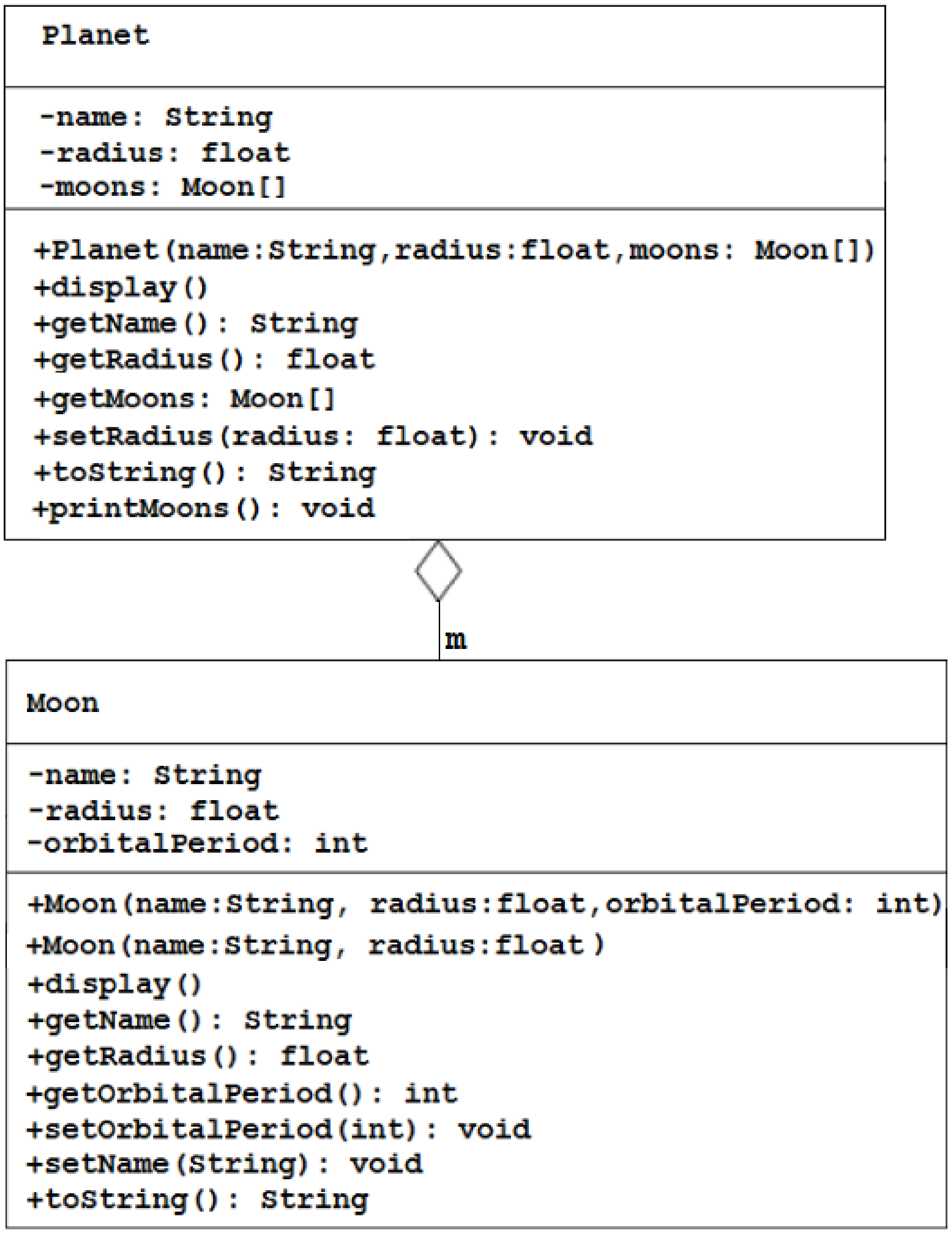
T - the type of objects that may be compared by this comparator

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.

Appendix C

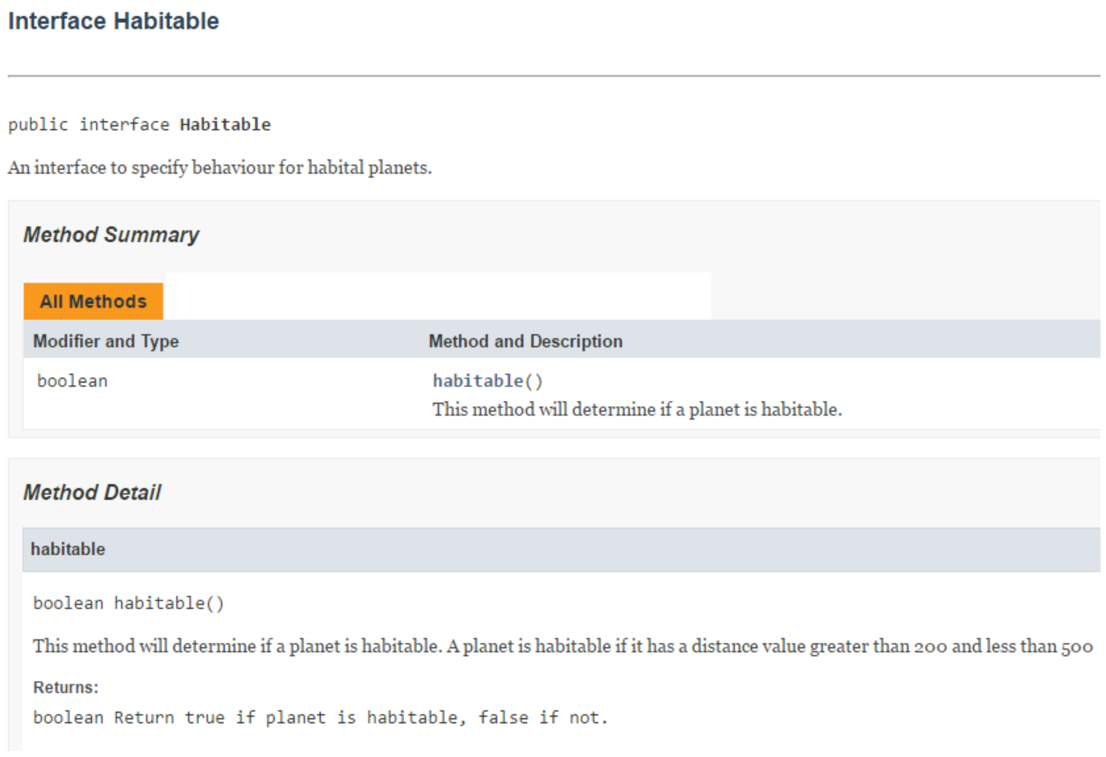

Planet
-name: String
-radius: float
-moons: Moon[]
+Planet(name:String,radius:float,moons: Moon[])
+display()
+getName(): String
+getRadius(): float
+getMoons: Moon[]
+setRadius(radius: float): void
+toString(): String
+printMoons(): void
m
Moon
-name: String
-radius: float
-orbitalPeriod: int
+Moon(name:String, radius:float,orbitalPeriod: int)
+Moon(name:String, radius:float)
+display()
+getName(): String
+getRadius(): float
+getOrbitalPeriod(): int
+setOrbitalPeriod(int): void
+setName(String): void
+toString(): String

Appendix D

(a)

```
public class Shape
{
  private int x;
  private int y;;

  public Shape(int x, int y)
  {
    this.x=x;
    this.y=y;
  }
  public void display()
  {
  }
}
```

(b)



Interface Habitable

```
public interface Habitable
```

An interface to specify behaviour for habital planets.

Method Summary

All Methods

Modifier and Type	Method and Description
`boolean`	`habitable()` This method will determine if a planet is habitable.

Method Detail

habitable

```
boolean habitable()
```

This method will determine if a planet is habitable. A planet is habitable if it has a distance value greater than 200 and less than 500

Returns:

`boolean Return true if planet is habitable, false if not.`

Appendix D continued

(c)

```
public class Planet
{
  public Planet()
  {
  }

}
```

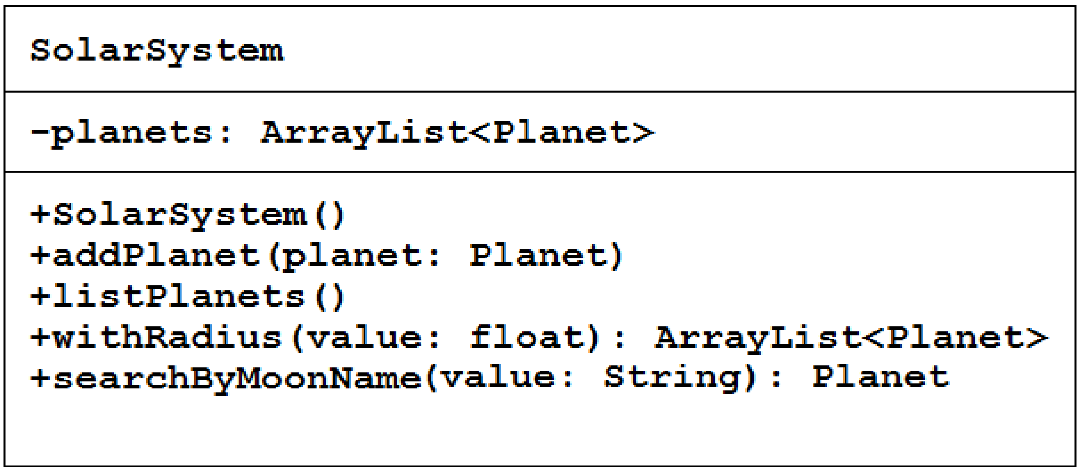
(d)

```
Planet planet = new Planet("earth",300);
if(. . .)
  println(planet.getName() + " is habitable");
```

(e)

```
public int binarySearch(int[] list, int searchItem)
{
  int mid=0;
  int start=0;
  int end=list.length-1;
  boolean found=false;

  //Loop until found or end of list.
  while (start <= end && !found)
  {
    mid = (start + end) / 2;
    if (list[mid] == searchItem)
    {
      found = true;
    }
    else
    {
      if (list[mid] > searchItem)
      {
        end = mid - 1;
      } else
      {
        start = mid + 1;
      }
    }
  }

  if(found)
    return mid;
  else
    return(-1);

}
```

Appendix E



```
SolarSystem
-------------------------------------------------------
-planets: ArrayList<Planet>
-------------------------------------------------------
+SolarSystem()
+addPlanet(planet: Planet)
+listPlanets()
+withRadius(value: float): ArrayList<Planet>
+searchByMoonName(value: String): Planet
```

`SolareSystem` uses `Planet` and `Moon` classes in Appendix C.

`SolarSystem` class tester code:

```
SolarSystem ss=new SolarSystem();

Moon[] earthMoon = new Moon[1];
earthMoon[0] = new Moon("Luna", 7.5, 28);
Planet earth = new Planet("earth", 25, earthMoon);
ss.addPlanet(earth);

Moon[] marsMoons = new Moon[2];
marsMoons[0] = new Moon("Phobos", 5, 28);
marsMoons[1] = new Moon("Deimos", 2.5, 42);
Planet mars = new Planet("mars", 20, marsMoons);
ss.addPlanet(mars);

ss.listPlanets();
println(ss.withRadius(25));
println(ss.searchByMoonName("Luna"));
```