Letterkenny Institute of Technology

Course code: OOPR CP603

YEAR 2 COMPUTING

(Common paper for all streams)

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

Date: January 2020 **Examiners:** Mr. T. Devine
Dr. L. Raeside

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 Pacman game that was introduced in the module.

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

(7 marks)

(b) Identify any shared class variable(s) and method(s) that both `Player` and `Enemy` classes use.

(2 marks)

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

(6 marks)

(d) Rewrite the signatures for the `Player` and `Enemy` classes to become subclasses of the `GameCharacter` class.

(2 marks)

(e) Write a new method for the `World` class called `printEnemies()` that will print the coordinates of each enemy character. The output should look like this if all enemies are at location (5,1). Use an enhanced for loop in your solution.

```
--Enemies--
[X:5 Y:1]
[X:5 Y:1]
[X:5 Y:1]
[X:5 Y:1]
```

(8 marks)

Question 2

Appendix B has a partial implementation of the `Player` class.

(a) Provide the code for a `toString()` method to print a string representation of the player object as shown in this example:

```
[Player: name=Joe Bloggs; age=21]
```

(6 marks)

(b) Provide the missing code for the `equals()` method.

(11 marks)

(c) Describe the purpose of the `@Override` annotation and give an appropriate example.

(4 marks)

(d) Describe the process of object casting in Java and give an appropriate example.

(4 marks)

Question 3

Appendix C (a) shows a class diagram for the `Player`, `Score`, `Goal` and `Point` classes. Provide all the Java code to implement only the `Player` class. Examine the tester code in *Appendix C (b)* to help implement the correct solution.

(25 marks)

Question 4

Examine the class diagram in *Appendix C (a)* and the `Score`, `Goal` and `Point` class code in *Appendix D (a)*.

(a) Identify the compilation error in the `Score` abstract class?

(2 marks)

(b) Rewrite the `Score` class to fix the error identified in (a).

(3 marks)

(c) Provide the code needed in `Goal` and `Point` to implement the `scoreValue()` abstract method. Note: a goal has a value of 3 and a point has a value of 1.

(8 marks)

Examine how the `Collections.sort()` method is used in *Appendix D (b)*.

(d) Provide the missing code for the class `ScoreComparator` that is used to sort players in ascending order by the *number of scores* they get.

(12 marks)

Question 5

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

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

(9 marks)

(c) *Appendix E* contains code for a selection sort. Complete the missing code in the method `selectionSort()`.

(4 marks)

(d) Use the binary search algorithm to find the value 30 in the list below. Clearly identify the values in `mid`, `start` and `end` during each pass.

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

(9 marks)

Question 6

Describe with code examples each of the following concepts:

(a) Accessor method
(b) Mutator method
(c) `try catch` statement
(d) `super` keyword
(e) `private` access modifier

(25 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)
  {
    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) - printEnemies()
  ...
  ...
}
```

Appendix A continued

```
//
// Player.java
//
public class Player // Q1 - (d)
{
  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 - (a)
     ...
     ...
     ...
    }

  }
}
```

Appendix A continued

```
//
// Enemy.java
//
public class Enemy // Q1 - (d)
{
  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;
    ...
    ...
    ...
    }
  }
}
```

Appendix A continued

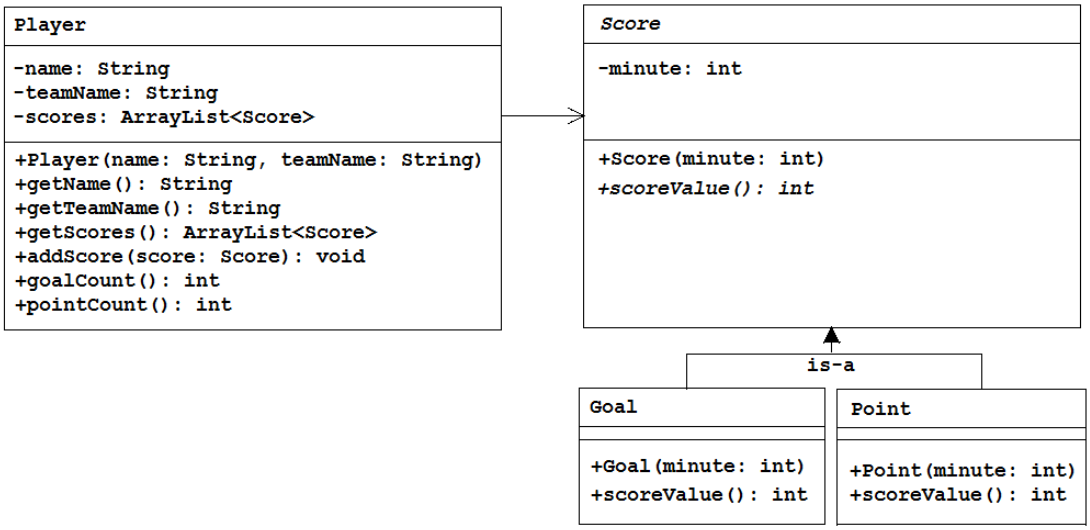
```
//
// Location.java
//
public class Location
{
  private int x;
  private int y;

  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

```
public class Player
{
  private String name;
  private int age;

  public Player(String name, int age)
  {
    this.name=name;
    this.age=age;
  }

  // Q2(a) – toString()


  // Q2(b) – equals()
  ...
  public ...(Object obj)
  {
    ... otherPlayer;
    if(obj instanceof ...)
      otherPlayer = (Player)obj;
    else
      return false;

    ...
    ...
    ...
  }
}
```

Appendix C

(a)



(b)

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

  Player p1 = new Player("M. Murphy","Donegal");

  p1.addScore(new Point(10)); // point scored @ 10th minute
  p1.addScore(new Point(22)); // point scored @ 22th minute
  p1.addScore(new Goal(29));  // goal scored @ 29th minute

  System.out.println(p1.getName());          // prints M. Murphy
  System.out.println(p1.getScores().size()); // prints 3
  System.out.println(p1.goalCount());        // prints 1
  System.out.println(p1.pointCount());       // prints 2

 }
}
```

Appendix D

(a)

```
public abstract class Score
{
  private int minute;

  public Score(int minute)
  {
    this.minute=minute;
  }

  public abstract int scoreValue()
  {
    return 0;
  }
}

public class Goal extends Score
{
  public Goal(int minute)
  {
    super(minute);
  }

}

public class Point extends Score
{
  public Point(int minute)
  {
    super(minute);
  }

}
```

Appendix D continued

(b)

```
import java.util.*;

public class Tester
{
  public static void main(String[] args)
  {
    Player p1 = new Player("M Murphy","Donegal");
    p1.addScore(new Point(10)); // point scored @ 10th minute
    p1.addScore(new Point(22)); // point scored @ 22th minute
    p1.addScore(new Point(29)); // point scored @ 29th minute

    Player p2 = new Player("P McBearty","Donegal");
    p2.addScore(new Point(12)); // point scored @ 12th minute
    p2.addScore(new Point(20)); // point scored @ 20th minute

    ArrayList<Player> players = new ArrayList<Player>();
    players.add(p1);
    players.add(p2);

    Collections.sort(players, new ScoreComparator());

  }
}


public class ScoreComparator ...
{
    @Override
    ...
    {
        ...
        ...
        ...
    }
}
```

Appendix E

```
public void selectionSort(int[] array)
{
  int temp; // temporary location for swap
  int max;  // index of maximum value in subarray
  for (int i=0;i<....length;i++)
  {
    // find index of largest value in subarray
    max=indexOfLargestElement(array, array.length-i);
    // swap
    temp=...;
    ...=array[array.length-i-1];
    array[array.length-i-1]=...;
  }
}

// Finds index of largest element
public int indexOfLargestElement(int[] array, int size)
{
  int index=0;
  for (int i=1; i<size; i++)
  {
    if (array[i]>array[index])
      index=i;
  }
  return index;
}
```