CSY2030 Inheritance – Interface and Abstract Classes Lab

1. Write an interface class called *Sport*. The interface should have methods for:

   getEquipment() which returns a string such as “Ball, Net”
   getName() which returns a string such as “Football” or “Tennis”
   *getLocation()*which returns a string such as “pitch” or “court”

2. Create an interface class called *TeamSport* which has methods for:

   getNumberOfTeams() which returns the number of teams in a match
   *getPlayersPerTeam()*which returns the number of players on each team

3. Create the following classes with the following information

Class Name	Interfaces	Attributes	Methods
BallSport	Sport, TeamSport	ObjectName (String) Equipment (String) Location (String) Number_of_teams (int) Players_Per_Team (in Time limit (int)	getTimeLimit();
MotorSport	Sport, TeamSport	ObjectName (String) Equipment (String) Location (String) Number_of_teams (int) Players_Per_Team (in Laps (int)	getNumberOfLaps()
RacketSport	Sport	ObjectName (String) Equipment (String) Location (String) Score_Limit (int)	getScoreLimit()
Golf	Sport	ObjectName (String) Equipment (String) Location (String)	

Remember to implement the interface methods as well (otherwise the sub classes won’t compile)

4. Using the classes from Q3, create objects for the following sports:

   Ball Sports:

Object Name	Equipment	Location	Number of teams	Players Per Team	Time limit
Football	Net, Ball	Pitch	2	11	90
Rugby	Goal, Ball	Pitch	2	15	90

   MotorSports:

Object Name	Equipment	Location	Number of teams	Players Per Team	#Laps
Formula 1	Car	Track	11	2	70
Moto GP	Motorcycle	Track	13	3	80

   RacketSports

Object Name	Equipment	Location	Score Limit
Tennis	Racket	Court	40
Badminton	Racket	Court	30

   Other

Object Name	Equipment	Location
Golf	Golf Clubs	Course

Store all your sport objects in a single array (you should store all the sports in a single array). Loop through them and display the information about each sport.

5. The code on NILE named *Exercise 8* contains the basics of a text based adventure game using *abstract* classes. Download and run the game.

   Now study the code – you will notice there are 6 java class files for this project:

   Weapon.java - this contains an abstract class which has the properties of a normal class (attributes, constructor and methods) and 2 abstract methods without code - it is up the subclasses to provide code for the abstract methods. Compare this abstract class to the interface classes you created in Q1 and Q2 of this lab

   Sword.java - this is a subclass of *Weapon* and uses *extends* (rather than *implements*) the superclass. Note that it provides the implementation for the abstract methods

   Dagger.java - this is also a subclass of *Weapon* and uses extends (rather than implements) the superclass. Note that it also provides the implementation for the abstract methods

   Enemy.java - this is a class to hold information about enemies

   Player.java - this is a class to hold information about players who are fighting enemies. Note that one of the attributes of Player is a *Weapon* object.

   Game.java - this holds the *main* method and is where the program is run. Look at the *inventory* array which is made up of a *Dagger* and *Sword* objects. Note that this array is passed into a method called *switchWeapon* which goes through the array and calls *getName()* for each element of the array i.e *inventory[i].getName()* - this is fine because this is defined in the Weapon class which Dagger and Sword objects can access.

   Also look at the method *block()* in *Game.java*:

```
public static int block(Player player, Enemy enemy) {
        return player.getWeapon().block(enemy);
}
```

   depending which type of Weapon (Dagger or Sword) that the method *getWeapon()* returns will determine which *block()* method is called. If getWeapon() returns a Dagger object it will call the *block()* method implemented in the Dagger class. If getWeapon() returns a Sword object it will call the *block()* method implemented in the Sword class. *Block()* is one of the abstract methods that Dagger and Sword had to implement.

   Similarly the *attack()* method in Game.java would call the *attack()* method in either Dagger or Sword.