Java Programming 1

PROFESSOR: CÂI FILIAULT

Topics for this week

- Objects continued
- Building and implementing methods
- Super and Sub classes
- Inheritance
- Method overriding and overloading

Record Games

NEW EMAIL

Email

Olivia

Company owner

Hello,

We would like to subcontract a feature of our game to your company.

We would like you to build a character, human, orc and elf class for our new upcoming hot title game. Please see the attached email for the specs of each class.

-see attached email

-Thanks

Email

Olivia

Company owner

Attached Email Part 1:

The following features are needed within the game:

The game will require the user to build a super class and 3 sub classes

- The super class will be the character class, as every orc, elf and human can be a character of a game. But the opposite is not true
- The orc, elf and human class will be sub classes of the character class.
- Each character can and will have the following attributes:
- Name, age, health, experience, intelligence, strength, accuracy, defense, wisdom, luck and agility
- Each race has a different spread of these attributes
- Each character should also be able to talk

Key Information

We need to build a super and sub class:

The **super class** being the **character** class and the **sub class** being **each of the 3 races**

Information about the class:

- The super class will require the user to build an assortment of attributes about the character.
- The super class should have getters and setters that are able to manipulate all the information
- Every character should be able to communicate. How they communicate depends on the race, if later we have a werewolf class they may communicate through howls, if we have a mutant class they may communicate through telepathy.
- We need to make a method that can be inherited by every sub class but not specify how it is used.

Key Information

Subclass:

- A subclass is a class that is derived from another class. In OOP subclasses are used to create more specific objects from something called a **base class.**
- Another name for a subclass is a child class. Classes can become subclasses by **extending** another class

Superclass:

- A superclass is the opposite of a subclass where other classes are derived from it. Another name of a superclass is a **base class** or a **parent class.**

Key Information

Extending a Class:

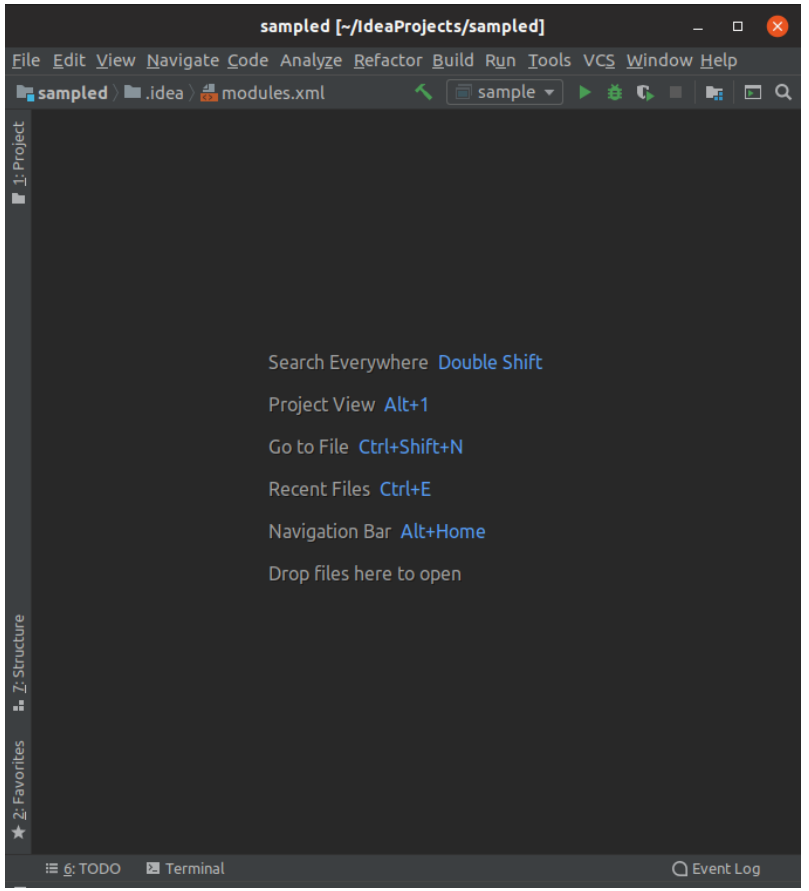
```
class student extends person {


}
```

When a class is extended from (in this case person), it becomes a superclass. The class that extends from it becomes a subclass (in this case student).

Creating a new project

Start by opening IntelliJ and selecting the following:

File> New> Project



New Project

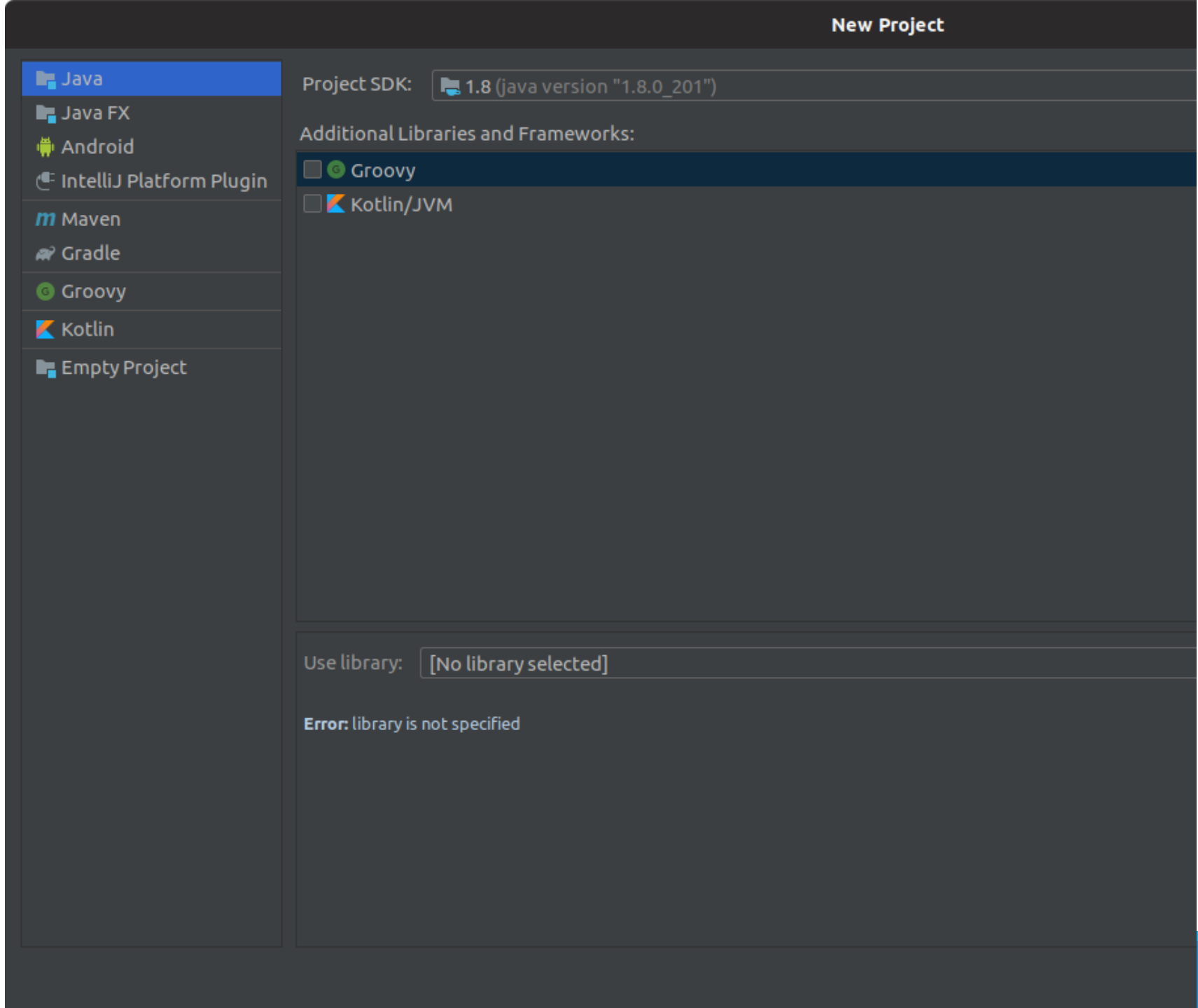
You will now see a dialog box appear *(It may be a little different than the one that I have shown)*

Starting on the left we will select a new Java Project

We will now want to select the project SDK of 1.8

SDK stands for Software Development Kit and it dictates which version of Java we are using.

At this point we can hit the next button twice



New Project

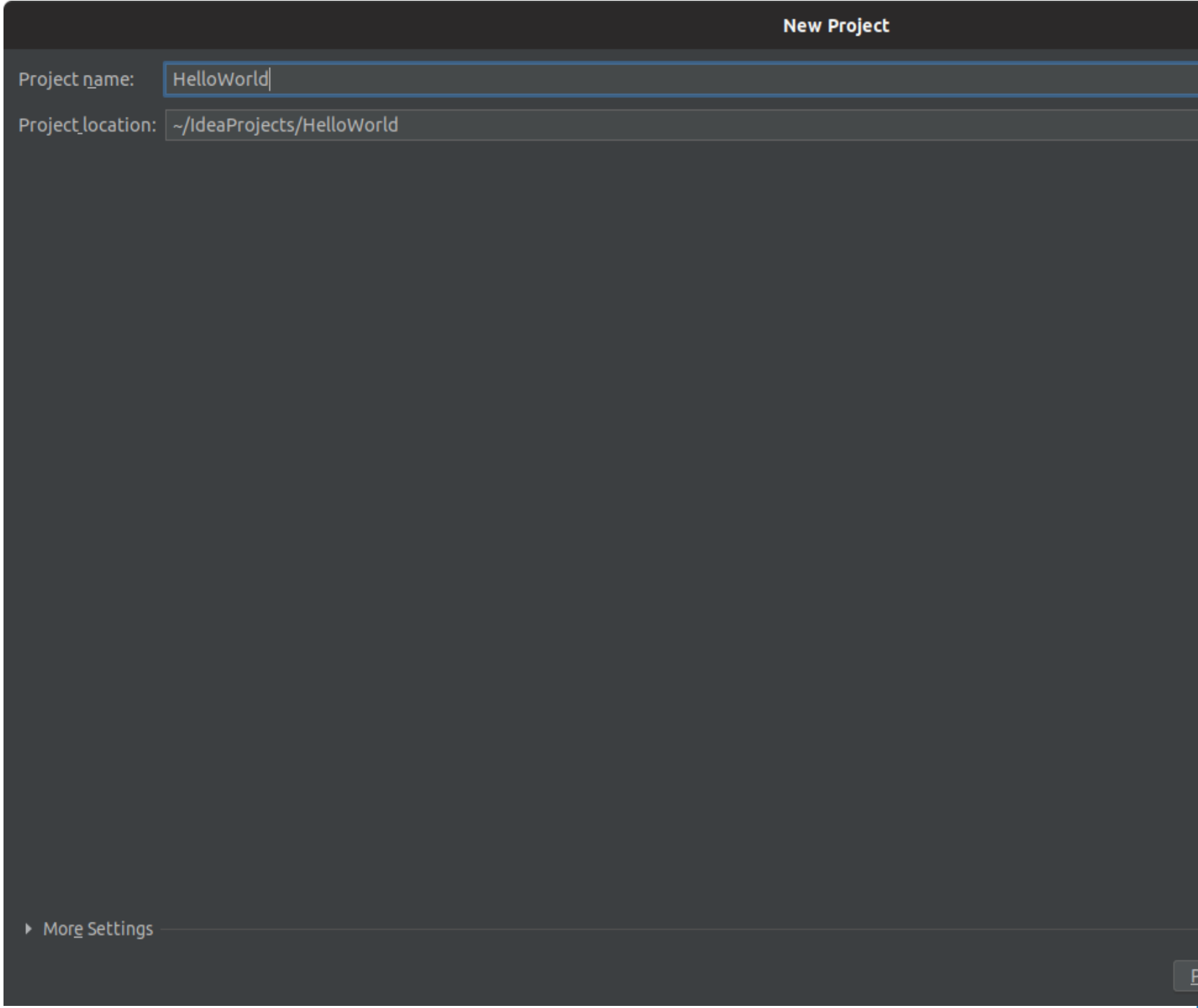
Give the project a name of:

Game

At this time you can change the project location to any convenient place you would like

Tip:

Keeping your programming projects on a flash/jump drive is perfectly fine. However you will want to avoid from directly working on the drive.

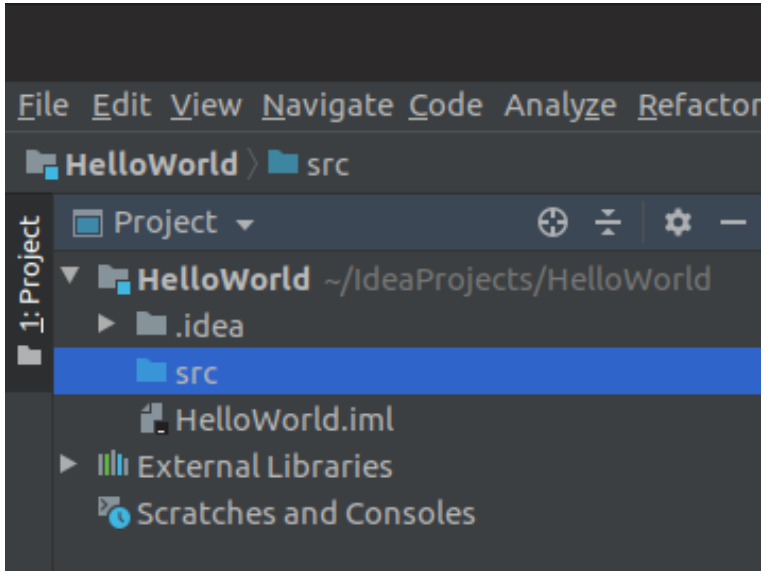


Creating a new project

You will now see in the top left-hand corner of your IDE a package explorer.

Within the package explorer you should see the **Game** project.

Expand the **Game** project to see a folder named **src**.



Creating a new class file

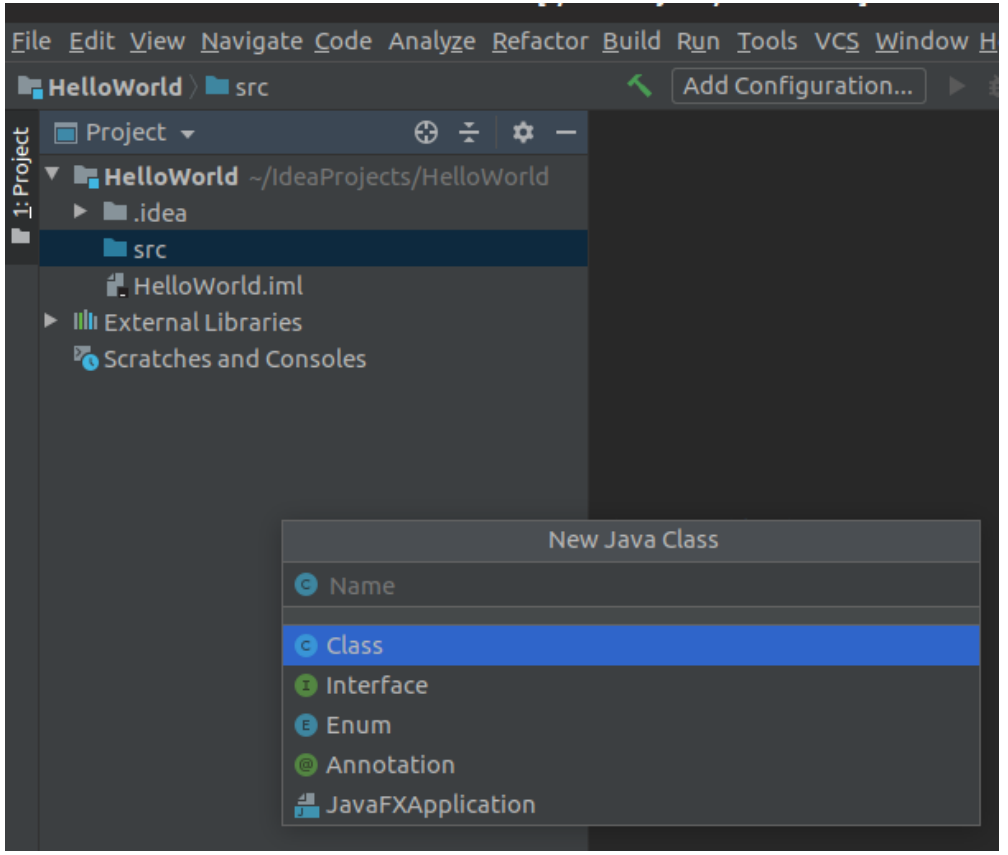
Next, we will need to create a new class file.

Right click on the **src** folder and select:

New> Java Class

Creating a new class is like creating a new program

Name the class **Character** and hit **enter**



Writing the program

- Let's start by creating member variables to store all the information about each Character
- This information should be initialized to default values
- The default values have been set to either 0 or an empty string because they will be instantiated by individual sub classes to the correct values
- We then create a character constructor which will be used to set the name of the character.
- This is the smallest piece of information needed to create a new character

```
1  package Games;
2
3  public class Character {
4
5      //create member data to store attributes of heros
6      private String name = "";
7      private int age = 0;
8      private int health = 0;
9      private int experience = 0;
10     private int intellegence = 0;
11     private int strength = 0;
12     private int accuracy = 0;
13     private int defense = 0;
14     private int wisdom = 0;
15     private int luck = 0;
16     private int agility = 0;
17
18
19     public Character(String name){
20         this.name = name;
21     }
22     //create getters and setters
23     //create a method to allow characters to speak
24
25 }
```

Writing the program

- Take a second to randomly assign some of the known attributes

```
public Character(String name) {
    this.name = name;
    Random r = new Random();
    this.setStrength(r.nextInt( bound: 100)+1);
    this.setAgility(r.nextInt( bound: 100)+1);
    this.setDefense(r.nextInt( bound: 100)+1);
    this.setIntelligence(r.nextInt( bound: 100)+1);
    this.setLuck(r.nextInt( bound: 100)+1);
    this.setAccuracy(r.nextInt( bound: 100)+1);
}
```

Writing the program

- We then create getters and setter for all the private information that was declared above
- We can see we have getters and setters for intelligence, strength, accuracy, defense, wisdom, luck and agility

```
20    public int getIntellegence() {
21        return intellegence;
22    }
23
24    public void setIntellegence(int intellegence) {
25        this.intellegence = intellegence;
26    }
27
28    public int getStrength() {
29        return strength;
30    }
31
32    public void setStrength(int strength) {
33        this.strength = strength;
34    }
35
36    public int getAccuracy() {
37        return accuracy;
38    }
39
40    public void setAccuracy(int accuracy) {
41        this.accuracy = accuracy;
42    }
43
44    public int getDefense() {
45        return defense;
46    }
47
48    public void setDefense(int defense) {
49        this.defense = defense;
50    }
```

Writing the program

- We then create more getters and setter for all the private information that was declared above
- We have age, experience, and name

```
52    public int getWisdom() {
53        return wisdom;
54    }
55
56    public void setWisdom(int wisdom) {
57        this.wisdom = wisdom;
58    }
59
60    public int getLuck() {
61        return luck;
62    }
63
64    public void setLuck(int luck) {
65        this.luck = luck;
66    }
67
68    public int getAgility() {
69        return agility;
70    }
71
72    public void setAgility(int agility) {
73        this.agility = agility;
74    }
75    public int getAge() {
76        return age;
77    }
78    public void setAge(int age) {
79        this.age = age;
80    }
81    public int getExperience() {
82        return experience;
83    }
```

Writing the program

```
87      public int getHealth() {
88          return health;
89      }
90      public void setHealth(int health) {
91          this.health = health;
92      }
93      public String getName() {
94          return name;
95      }
96      public void setName(String name) {
97          this.name = name;
98      }
```

Writing the program

- The next issue that we need to overcome is creating a method that allows the character to communicate
- We may have many different races and they may communicate in different ways.
- Some may use sign language, some use their mouth, others use telepathy. This is an abstract concept and calls for the use of an abstract method.
- An abstract method is a method that no definition is given
- An abstract method is more of a contract to any class that extends (or is a sub class) of the class we have created
- The contract states that any class that extends this class must create its own communicate method
- When we create an abstract method, we must also make the class abstract.

```
public abstract void speak();
```

Please note in the lecture we replaced speak() with communicate().
Please refer to the code solutions on blackboard for the exact code we used

Writing the program

- When a class is an abstract class it means no objects can be created from it.
- The idea is abstract classes can contain abstract methods
- And because these abstract methods do not have a definition, we need to ensure that there are no “character” objects that accidently use these abstract methods
- Ultimately leading to a catastrophic failure in our program
- Instead we say that an abstract class is a class that can only every be a parent or super class.
- Any class that extends an abstract class must meet all the contract requirements and may create objects of its own.
- In this case we might want to add universal functionality to each of our races but need to follow a concept that a character will ALWAYS have a race.

```
3  public abstract class Character {
```

Creating a new class file

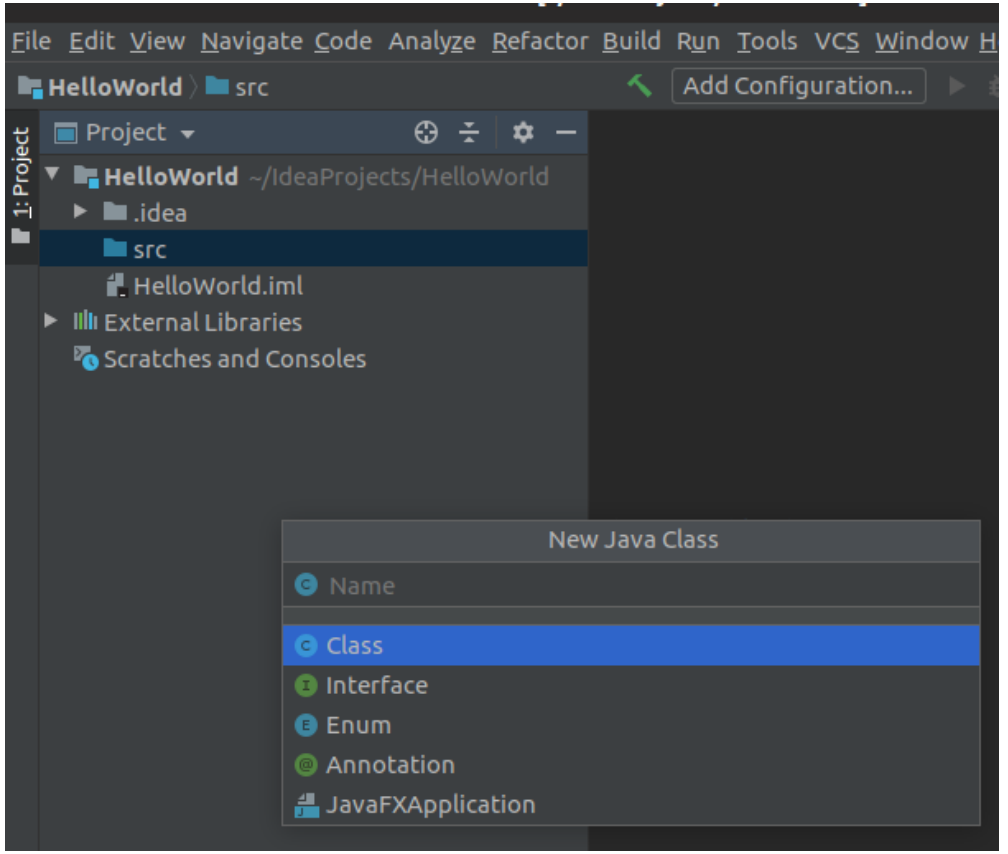
Next, we will need to create a new class file.

Right click on the **src** folder and select:

New> Java Class

Creating a new class is like creating a new program

Name the class **Human** and hit **enter**



Writing the program

Let's create a constructor for the human class that sets all the appropriate numbers for each of the attributes

We then create a communicate method as part of the contract when we extend the class Character.

```
public class Human extends Character{
    public Human(String name) {
        super(name);
        this.setMortal(true);
        this.setMagical(false);
        this.setAge(20);
        Random r = new Random();
        this.setHealth(r.nextInt( bound: 100)+1);
    }

    @Override
    public void speak() {
        System.out.println("Hello I am a human and my name is "
                + this.getName());
    }

    @Override
    public String toString() {
        return this.getName() + " Mortal: "
                + isMortal() + " Magical: " + isMagical();
    }
}
```

Creating a new class file

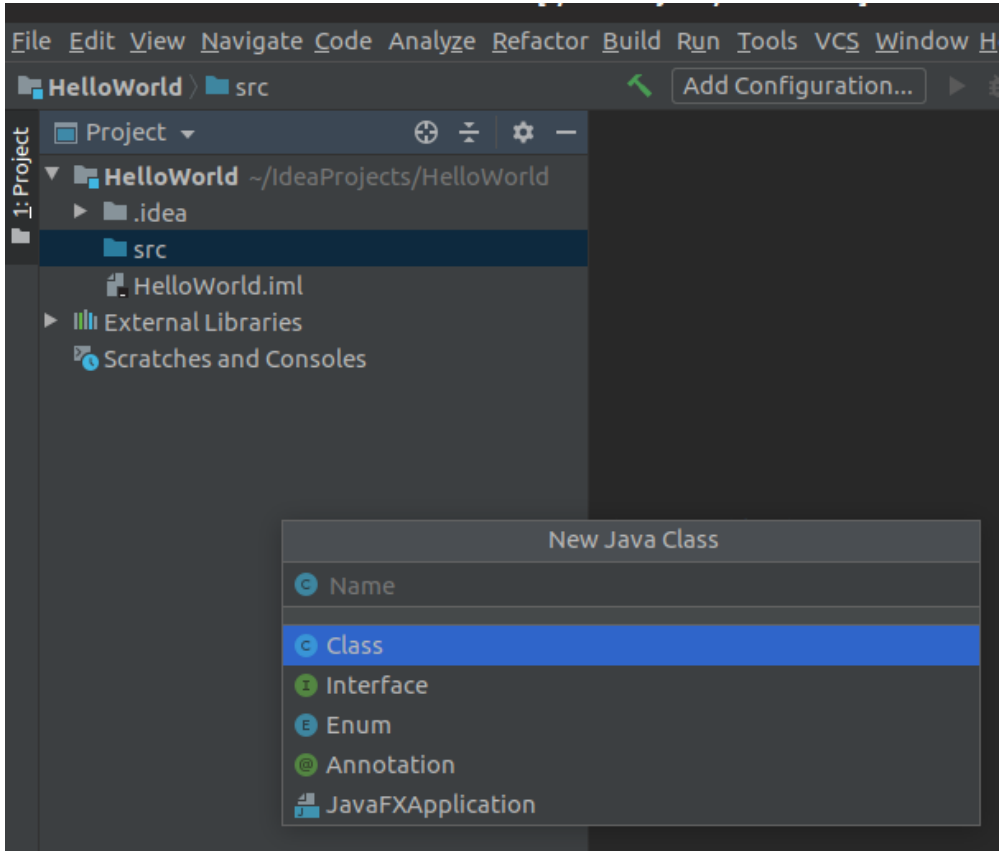
Next, we will need to create a new class file.

Right click on the **src** folder and select:

New> Java Class

Creating a new class is like creating a new program

Name the class **Orc** and hit **enter**



Writing the program

Let's create a constructor for the orc class that sets all the appropriate numbers for each of the attributes

We then create a communicate method as part of the contract when we extend the class Character.

```
public class Orc extends Character{
    boolean hasTusk = true;
    public Orc(String name) {
        super(name);
        this.setMortal(false);
        this.setMagical(false);
        Random r = new Random();
        this.setAge(220);
        this.setHealth(r.nextInt( bound: 250)+1);
    }

    public boolean isHasTusk() {
        return hasTusk;
    }

    public void setHasTusk(boolean hasTusk) {
        this.hasTusk = hasTusk;
    }

    @Override
    public void speak() {
        System.out.println("I am an orc");
    }
}
```

Creating a new class file

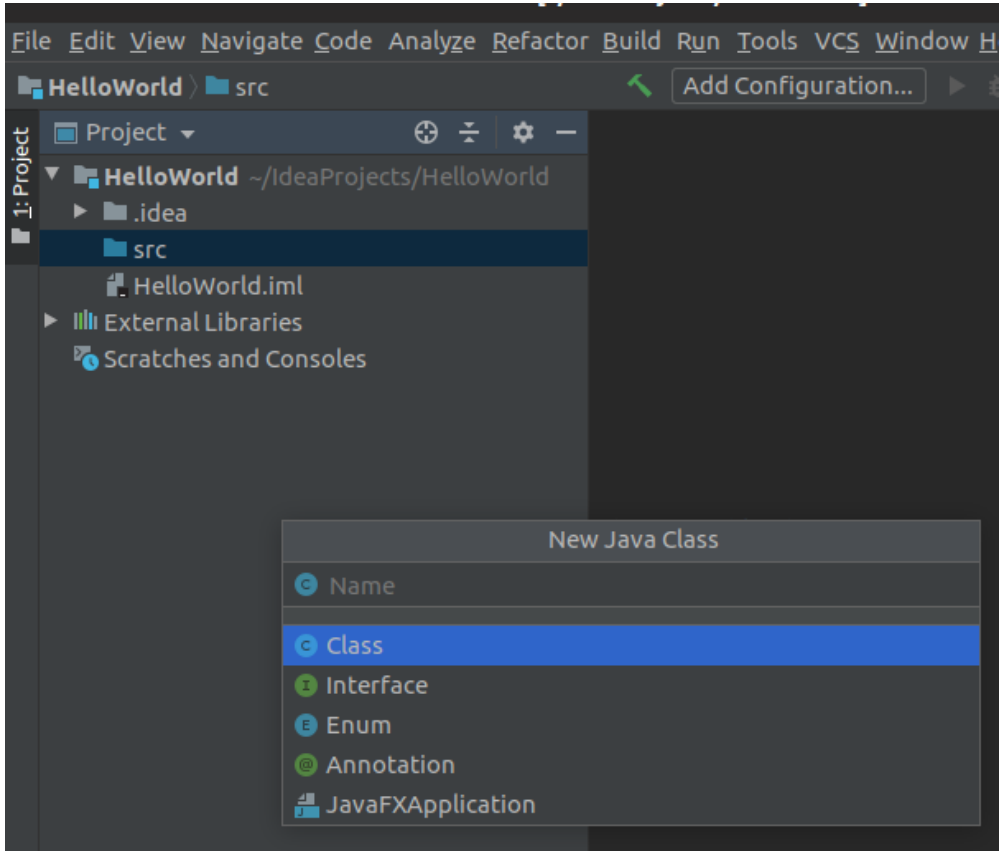
Next, we will need to create a new class file.

Right click on the **src** folder and select:

New> Java Class

Creating a new class is like creating a new program

Name the class **Elf** and hit **enter**



Writing the program

Let's create a constructor for the elf class that sets all the appropriate numbers for each of the attributes

We then create a communicate method as part of the contract when we extend the class Character.

```
public class Elf extends Character {
    boolean pointedEars = true;
    public Elf(String name) {
        super(name);
        this.setMortal(false);
        this.setMagical(true);
        Random r = new Random();
        this.setAge(1020);
        this.setHealth(r.nextInt( bound: 1000)+1);
    }
    public boolean isPointedEars() {
        return pointedEars;
    }
    public void setPointedEars(boolean pointedEars) {
        this.pointedEars = pointedEars;
    }
    @Override
    public void speak() {
        System.out.println("I am an elf");
    }
}
```

Writing the program

- The game class creates an elf, orc and human object
- We then address another new concept
- Let's look at how we create an array of characters
- Now initially when we disussed arrays we had mentioned that they could only be of primitive data types such as int, Boolean, float, double, char, string etc.
- We can also create arrays that are of an object type
- Just like we discussed last week with variables
- We said we could create variables that are of object types and are able to store objects within them
- Note how we create an array that can store characters. An elf, orc and human are all characters. They are just a subclass of characters. Therefore can store them all in an array of characters
- We then universally use the communicate method on each character in the characters array

```
package Game;

public class Game {

    public static void main(String[] args) {

        Elf character1 = new Elf("Harry");
        Orc character2 = new Orc("Brad");
        Human character3 = new Human("Jen");
        Character[] list = new Character[3];
        list[0] = character1;
        list[1] = character2;
        list[2] = character3;

        for(int i = 0; i < list.length; i++){
            list[i].speak();
        }
    }

}
```

Test your software

Take a moment to test your software

Homework

Read chapter 11 of your textbook

Next Week

Test