MODIFIERS

Scenario

A **Bowl** HAS-A **Snake** (aggregation). As the temperature of the **Bowl** increases, the height of the **Snake** increases. The maximum height the Snake can reach is 120cm. This is a FIXED value and cannot be changed.

If the **Snake** reaches this maximum height, it means the temperature of the **Bowl** is very high and the **Bowl** cracks and the **Snake** escapes.

Draw UML diagrams showing the relationship between the Bowl and Snake classes. You do not have to include attributes or methods, only the class title. Make sure each diagram has 3 tiers and a line demonstrating the relationship between each class.

```
package bowlsnakepackage

public class Snake{

  private final int maxHeight = 120;

  private int currentHeight;

  public Snake(){

    this.currentHeight = 0;

  }

  public int getCurrentHeight(){

    return this.currentHeight;

  }

  public int getMaxHeight(){

    return this.maxHeight

  }

  public void setCurrentHeight(int newHeight){

    this.currentHeight = newHeight;

  }

}
```

Here is the Snake class:

i. Circle the constructor method

ii. How many public **modifiers** are there?

iii. Underline an **attribute** whose value cannot be changed.

iv. Explain your answer to iii:

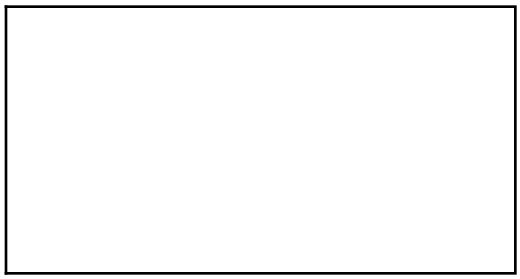
Notice that the Snake class is a member of a package. In fact, as you will see, all classes in this project are members of the same package.

Here is the **Bowl** class:

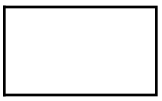
```
package bowlsnakepackage

public class Bowl{

  private int bowlTemp;

  private Snake snake;

  private boolean isCracked;

  public Bowl(Snake snake){

    this.snake = snake;

    this.bowlTemp = 20;

    this.isCracked = false;

  }

  public int getBowlTemp(){

    return this.bowlTemp;

  }

  public void setBowlTemp(){

    //set the bowl's temp between 20 and 120

    bowlTemp = (int)(Math.random()*101)+20;

    //move the Snake according to the temperature

    //update the bowl's cracked status

    moveSnake();

    checkCracked();

  }

  private void moveSnake(){

    snake.setCurrentHeight(bowlTemp);

  }

  private void checkCracked(){

    if(bowlTemp >= 120){

      this.isCracked = true;

    }

  }

  public boolean getCracked(){

    return this.isCracked;

  }

}
```

i. circle a mutator method

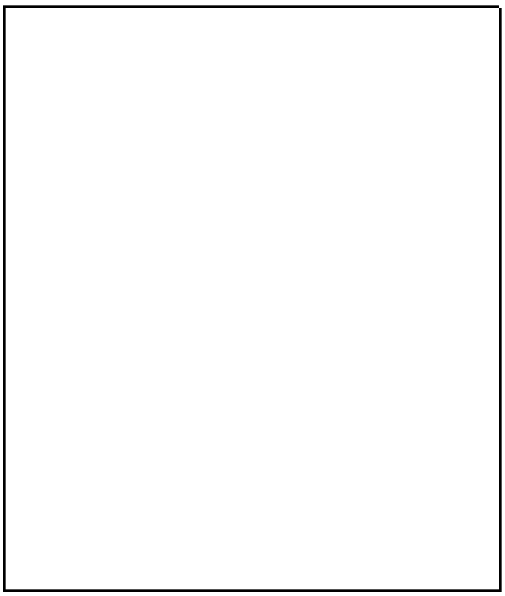
ii. Describe the purpose of a mutator method



ii. how many methods use the private **modifier**?



iii. why do these methods use the private **modifier** and not the public modifier?



The Bowl class is also a member of the **bowlsnakepackage** package.

Classes in the same package should trust each other!

So, instead of making their attributes **private**, they could/should use the **protected** modifier. This means classes in the same package will be able to see the data in the attributes of other package classes. There will be no need to call accessor/mutator methods.

Classes outside the package will still need to use accessor and mutator methods for attributes using the **protected** modifier.

Adapt the **Snake** class by making the 2 attributes **protected**:

Now, adapt the `moveSnake` method of the **Bowl** class. How can we change it to take advantage of the **protected** modifier of **Snake** class attributes. Rewrite the method here:

So far we have become familiar with 2 modifiers: **public** and **private**

This unit has introduced 2 more: **final** and **protected**.

A variety of modifiers are available to designers of computer programs and the more you become familiar with them, the more effective your computer programs will become.

Programming Task #1

Open your IDE and create the Snake and Bowl class inside a package called **bowlsnakepackage**.

Rewrite the 2 classes to take advantage of the **protected** modifier.

Now design a Main.java file which constructs an instance of a **Bowl** which HAS-A **Snake**.

Alter the temperature of the **Bowl** until it cracks and the **Snake** escapes.

What questions do you currently have about the concepts offered in this unit? Write your questions here:

Programming Task #2

Another modifier commonly used in Java is the **static** modifier which you have seen many times.

But what is the meaning of this modifier?

A property or method which uses the **static** modifier **belongs to the class** and not to an instance of the class!

For example, let's say that the Snake class wants to track how many instances of Snakes are being constructed in a computer program. Within the Snake class, we can use a static property to help out:

```
private static snakeCount = 0;
```

Then in the constructor method we can increment this count:

```
public Snake(){
      snakeCount = snakeCount + 1;
}
```

Now we can construct a Snake instance in another class:

```
Snake s1 = new Snake();
System.out.println(s1.getSnakeCount());
```

Oh - we forgot to design an **accessor** method for this new property! So, let's do it:

```
public int getSnakeCount(){
      return this.snakeCount;
}
```

Try it! It won't work.

The property is static. It belongs to the class, not an instance of the class. So, we can't access this property via an object reference variable eg `s1.getSnakeCount()`

We can access it only as a class variable... so...

```
System.out.println(Snake.getSnakeCount());
```

Oh! So, the accessor method needs to be **static** too:

```
public static int getSnakeCount(){
        return this.snakeCount;
}
```

Does it work?!

NO!!

The keyword **this** refers to **this instance of** a Snake. But snakeCount is not an instance variable, it is a class variable. And so, you cannot use the **this** keyword with static variables in Java!

```
public static int getSnakeCount(){
        return snakeCount;
}
```

Does it work?!

Take some time to think about it.

Before when you were designing modular programs with **helper functions**, you always used the **static** keyword eg

```
public static int doubleIt(int x){
        return x*2;
}
```

And when your program wanted to call the function, we didn't need a reference variable to call the method/function:

```
public static void main(String[ ] args){
      doubleIt(3);
}
```

Also, **global** variables were **static**, meaning that they were not tied to a single instance/object:

```
static int population;
```