# Evidence for Implementation and Testing Unit.

Ryan Mackay

E20

**I.T 1- Demonstrate one example of encapsulation that you have written in a program.**

```java
public abstract class VideoDevice {
    private int screenSize;
    private int pixels;

    public VideoDevice(int screenSize, int pixels) {
        this.screenSize = screenSize;
        this.pixels = pixels;
    }

    public int getScreenSize() { return this.screenSize; }

    public int getPixels() { return this.pixels; }

    public String display(String data) { return data + " is now on screen"; }
}
```

**I.T 2 - Example the use of inheritance in a program.**

- **A class**

```java
public class Person {

    private String name;
    private String cohort;

    public Person(String name, String cohort) {
        this.name = name;
        this.cohort = cohort;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getCohort() { return cohort; }

    public void setCohort(String cohort) { this.cohort = cohort; }

    public String talk(String language) { return "I love " + language; }
}
```

- **A class that inherits from another class**

```java
public class Instructor extends Person{

    private String moduleTeam;

    public Instructor(String name, String cohort, String moduleTeam) {
        super(name, cohort);
        this.moduleTeam = moduleTeam;
    }


}
```

- **An object in the inherited class**

```java
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class InstructorTest {

    Person instructor;

    @Before
    public void before() { instructor = new Person( name: "Ally",  cohort: "G3"); }


    @Test
    public void hasName() { assertEquals( expected: "Ally", instructor.getName()); }

    @Test
    public void hasCohort() { assertEquals( expected: "G3", instructor.getCohort()); }

    @Test
    public void canChangeName(){
        instructor.setName("Sandy");
        assertEquals( expected: "Sandy", instructor.getName());
    }
    @Test
    public void canChangeCohort(){
        instructor.setCohort("G4");
        assertEquals( expected: "G4", instructor.getCohort());
    }

    @Test
    public void canTalk() { assertEquals( expected: "I love Java", instructor.talk( language: "Java")); }


}
```

- **A method that uses the information inherited from another class**

```java
@Test
public void hasName() { assertEquals( expected: "Ally", instructor.getName()); }
```

**I.T 3 - Example of searching**

```
def self.find( id )
  sql = "SELECT * FROM pizza_orders WHERE id = $1"
  values = [id]
  pizza = SqlRunner.run( sql, values )
  result = PizzaOrder.new( pizza.first )
  return result
end
```

Result of the function running (searched id 5)

```
#<PizzaOrder:0x007f89bd0e41f8 @id=5, @first_name="Ryan", @last_name="Mackay", @topping="Margherita", @quantity=9002, @price=10>
```

**I.T 4 – Example of sorting**

**I.T 5 -**

**Example of an array**

```
@stops = [ "Croy", "Cumbernauld", "Falkirk High", "Linlithgow", "Livingston", "Haymarket" ]
```

**a function that uses an array**

```
def add_stop(new_stop)
  @stops.push(new_stop)
  p @stops
end



add_stop("Edinburgh")
```

**the result**

```
➜  day_3 ruby IT_5.rb
["Croy", "Cumbernauld", "Falkirk High", "Linlithgow", "Livingston", "Haymarket", "Edinburgh"]
➜  day_3
```

**I.T 6 - Example of a hash**

```
@countries = {
  uk: {
    capital: "London",
  population: 1000
  },
  germany: {
  capital: "Berlin",
  population: 5
  }
}
```

**a function that uses a hash**

```
def update_population()
  @countries[:germany] [:population] = 8000
  p @countries[:germany]
end
```

**and the result**

```
{:capital=>"Berlin", :population=>8000}
```

**I.T 7 - Example of polymorphism in a program**

```java
public interface IRestrictable {

    boolean isAllowedTo(Visitor visitor);

}
```

```java
public abstract class Rollercoaster extends Attraction implements IChargeable, IRestrictable {


    public Rollercoaster(String name, int funLevel) {
        super(name, funLevel);
    }

    @Override
    public boolean isAllowedTo(Visitor visitor){
        boolean isOldEnough = visitor.getAge() >= 12;
        boolean isTallEnough = visitor.getHeight() >= 145;
        return isTallEnough && isOldEnough;
    }

    }
```