Evidence for Implementation and Testing Unit

Ross Hill
Cohort E18

I.T. 1 – Encapsulation in a program

```ruby
class Rooms

attr_reader :room_name, :room_capacity, :room_fee, :songs_in_room

  def initialize (room_name, room_capacity, room_fee, songs_in_room)
    @room_name = room_name
    @guests_in_room = []
    @songs_in_room = []
    @room_capacity = room_capacity
    @room_fee = room_fee
  end

  def guest_count
    @guests_in_room.length()
  end

  def add_guest_to_room(guest_name)
    @guests_in_room << guest_name
  end

  def remove_guest_from_room(guest_name)
    @guests_in_room.delete(guest_name)
  end

  def room_capacity_check
    if guest_count > @room_capacity
      return "Room full!"
    else
      return guest_count
    end
  end
```

I.T. 2 – Inheritance in a program

```java
package Player.Fighters;
import Actions.IAttackable;
import NPC.Enemy;

public abstract class Fighters extends Player.Player implements IAttackable {

    private Weapon weapon;

    public Fighters(String name, int healthPoints) {
        super(name, healthPoints);
        this.weapon = null;
    }

    public void setWeapon(Weapon weapon) {
        this.weapon = weapon;
    }

    public void attack(Enemy enemy) {
        int enemyHealth = enemy.getHealth();
        int weaponValue = weapon.getValue();
        enemy.setHealth(enemyHealth - weaponValue);
    }
}
```

```java
package Player.Fighters;


public class Knight extends Fighters {


    public Knight(String name, int healthPoints){
        super(name, healthPoints);
    }


}
```

```java
import NPC.Orc;
import Player.Fighters.Barbarian;
import Player.Fighters.Dwarf;
import Player.Fighters.Knight;
import Player.Fighters.Weapon;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

Run Test | Debug Test
public class PlayerTest {

    Knight knight;
    Barbarian barbarian;
    Dwarf dwarf;

    @Before
    public void before(){
        knight = new Knight("Arthur", 400);
        barbarian = new Barbarian("Conan", 500);
        dwarf = new Dwarf("Gimmly", 350);
    }

    @Test
    Run Test | Debug Test
    public void hasName() {
        assertEquals("Arthur", knight.getName());
        assertEquals("Conan", barbarian.getName());
        assertEquals("Gimmly", dwarf.getName());
    }

    @Test
    Run Test | Debug Test
    public void knightCanAttackOrc() {
        Orc orc = new Orc("Mr. Orc", 400);
        knight.setWeapon(Weapon.SWORD);
        knight.attack(orc);
        assertEquals(350, orc.getHealth());
```

I.T. 3 – Searching in a program

```ruby
def Artist.find(id)
  sql = "SELECT * FROM artists WHERE id = $1"
  result = SqlRunner.run(sql, [id])
  return result.map {|artist| Artist.new(artist)}
end
```

```
[7] pry(main)> Artist.find(2)
=> [#<Artist:0x007f9ad7aa3768 @id=2, @name="Hot Snakes">]
[8] pry(main)>
```

I.T. 4 – Sorting data in a program

examples.rb

```ruby
birds = ["sparrow", "robin", "pigeon", "magpie", "penguin"]

def sort_A_to_Z(array)
  p array.sort
end

sort_A_to_Z(birds)
```

```
→  pda git:(master) ✗ ruby examples.rb
["magpie", "penguin", "pigeon", "robin", "sparrow"]
→  pda git:(master) ✗
```

I.T. 5 – Use of an array

```ruby
fruits = ['banana', 'apple', 'orange']
```

```ruby
def add_fruit(new_fruit)
  fruits = ['banana', 'apple', 'orange']
  fruits.push(new_fruit)
  p fruits
end

add_fruit('avocado')
```

```
→  pda git:(master) ✗ ruby examples.rb
["banana", "apple", "orange", "avocado"]
→  pda git:(master) ✗
```

I.T. 6 – Use of a hash

```ruby
@customer =
  {
    name: "Ross",
    pets: [],
    cash: 1000
  }

def add_cash(add_amount)
  @customer[:cash] += add_amount
  p @customer
end

add_cash(800)
```

```
→  pda git:(master) ✗ ruby examples.rb
{:name=>"Ross", :pets=>[], :cash=>1800}
→  pda git:(master) ✗
```

I.T. 7 – Use of Polymorphism

```
1    public interface IPlayable {
2
3     String play();
4    }
5
6
7
8
```

```
1   public abstract class Instrument {
2
3       String colour;
4       int neighbour_annoyance_level;
5
6       public Instrument(String colour, int neighbour_annoyance_level) {
7           this.colour = colour;
8           this.neighbour_annoyance_level = neighbour_annoyance_level;
9       }
10
11      public String getColour() {
12          return colour;
13      }
14
15      public int getNeighbour_annoyance_level() {
16          return neighbour_annoyance_level;
17      }
18  }
19
```

```java
public class Guitar  extends Instrument implements IPlayable{

    private int no_of_strings;
    private int pickups;
    private String fretboard_material;

    public Guitar(String colour, int neighbour_annoyance_level, int no_of_strings, int pickups, String fretboard_material) {
        super(colour, neighbour_annoyance_level);
        this.no_of_strings = no_of_strings;
        this.pickups = pickups;
        this.fretboard_material = fretboard_material;

    }

    public String play() {
        return "RIFFFFFFFS";
    }

    public int getNoOfStrings() {
        return no_of_strings;
    }

    public int getNoOfPickups() {
        return pickups;
    }

    public String getMaterial() {
        return fretboard_material;
    }
}
```

```java
public class Piano extends Instrument implements IPlayable{

    private int no_of_keys;
    private String type;

    public Piano(String colour, int neighbour_annoyance_level, int no_of_keys, String type) {
        super(colour, neighbour_annoyance_level);
        this.no_of_keys = no_of_keys;
        this.type = type;
    }

    public String play() {
        return "TINKLING OF KEYS";
    }

    public int getKeys() {
        return no_of_keys;
    }

    public String getType() {
        return type;
    }
}
```