Implementation and Testing Unit
SQA PDA: Software Development
Yan Ren

IT1 - Encapsulation

```ruby
require_relative('../db/sql_runner')

class Company

  attr_reader :id, :name, :location, :logo

  def initialize(options)
    @name = options["name"]
    @location = options["location"]
    @logo = options["logo"]
    @id = options["id"].to_i if options["id"]
  end

  def save()
    sql = "INSERT INTO companies (name, location, logo) VALUES ('#{@name}','#{@location}', '
      #{@logo}') RETURNING id;"
    result = SqlRunner.run(sql)
    @id = result[0]["id"].to_i
  end

  def update()
    sql = "UPDATE companies
          SET name = '#{@name}',
          location = '#{@location}',
          logo = '#{@logo}'
          WHERE id = #{@id};"
    return SqlRunner.run(sql)
  end

  def delete()
    sql = "DELETE FROM companies WHERE id = #{@id};"
    return SqlRunner.run(sql)
  end

  def self.all()
    sql = "SELECT * FROM companies;"
    companies = SqlRunner.run(sql)
    return companies.map {|company| Company.new(company)}
  end
```

IT2 - Inheritance

```java
package music_shop;

public abstract class StringInstrument {

  int numOfStrings;
  String brand;


  public StringInstrument(int numOfStrings, String brand) {
    this.numOfStrings = numOfStrings;
    this.brand = brand;


  }
}
```

```java
package music_shop;
import behaviours.*;
import sellable.*;

public class Violin extends StringInstrument implements Playable, Sellable {


  public Violin(int numOfStrings, String brand){
    super(numOfStrings, brand);

  }

  public int getNumStrings() {
    return numOfStrings;
  }

  public String getBrand() {
    return brand;
  }

  public String play() {
    return "Violin Plays!";
  }

}
```

```java
import static org.junit.Assert.*;
import org.junit.*;
import music_shop.*;
import behaviours.*;

public class ViolinTest {

  Violin violin;


  @Before
  public void before(){
    violin = new Violin(5, "Yamaha");

  }

  @Test
  public void hasNumStrings() {
    assertEquals(5, violin.getNumStrings());
  }


  @Test
  public void canPlay() {
    assertEquals("Violin Plays!", violin.play());

  }

}
```

IT3 - Data Searching

```ruby
def FindingOddNumbers (numbers)
  result = numbers.find_all{ |n| n % 2 == 1}
  puts result
end

numbers = [3, 7, 8, 9]
FindingOddNumbers(numbers)
```

```
PDA_evidence  PDA_materials search.rb      week_3
➜  pda ruby search.rb
3
7
9
➜  pda
```

IT4 - Data Sorting

```ruby
def sortNumbers(numbers)
  numbers.sort!
  puts numbers
end

numbers = [4, 6, 3, 4, 9, 30]
sortNumbers(numbers)
```

```
➜  pda ruby sort.rb
3
4
4
6
9
30
➜  pda
```

IT5 - Array

```ruby
def reduce_number_in_array(array, number)

    result = array.map {|x| x - number}
    puts result

end

array1 = [3, 4, 5, 6]
number1 = 1

reduce_number_in_array(array1, number1);
```

```
➜  pda ruby array.rb
2
3
4
5
➜  pda
```

IT6 - Hash

```ruby
def search_and_delete(hash, number)
  result = hash.delete_if{|key,value| value <= number}
  puts result
end

hash1 = {"apple"=>10, "pear"=>30, "banana"=>15}
number1 = 15

search_and_delete(hash1, number1)
```

```
➜  pda ruby hash.rb
{"pear"=>30}
➜  pda
```

IT7 - Polymorphism

```java
class Overload{

    public int add(int x, int y){  //method 1

    return x+y;

    }

    public int add(int x, int y, int z){ //method 2

    return x+y+z;

    }

}

class Test{

    public static void main(String[] args){

    Overload demo = new Overload();

    System.out.println(demo.add(2,3));      //method 1 called

    System.out.println(demo.add(2,3,4));    //method 2 called


    }

}
```