

## Evidence for the PDA in software development I&T Implementation and Testing Unit

Benjamin Bowen  
Cohort E17

IT 1

Take a screenshot of an example of encapsulation in a program.

```
public class Room {  
    private String name;  
    private ArrayList<Guest> guests;  
    private int capacity;  
    private double price;  
    private boolean exclusive;  
  
    public Room(String name, int capacity, double price, boolean exclusive) {  
        this.name = name;  
        this.guests = new ArrayList<>();  
        this.capacity = capacity;  
        this.price = price;  
        this.exclusive = exclusive;  
    }  
  
    public int getCapacity() { return this.capacity; }  
  
    public double getPrice() { return this.price; }
```

The fields for the class are private so cannot be accessed directly, but there are methods that allow them to be accessed.

IT 2

Take a screenshot of the use of Inheritance in a program. Take screenshots of:

- A Class

```
public class Room {  
    private String name;  
    private ArrayList<Guest> guests;  
    private int capacity;  
    private double price;  
    private boolean exclusive;  
  
    public Room(String name, int capacity, double price, boolean exclusive) {  
        this.name = name;  
        this.guests = new ArrayList<>();  
        this.capacity = capacity;  
        this.price = price;  
        this.exclusive = exclusive;  
    }  
  
    public int getCapacity() { return this.capacity; }  
  
    public double getPrice() { return this.price; }  
  
    public ArrayList<Guest> getGuests() { return guests; }  
  
    public int getNumberOfGuests() { return this.guests.size(); }  
  
    public boolean roomIsEmpty() { return this.getNumberOfGuests() == 0; }  
  
    public void addGroupToRoom(Group group) { this.guests.addAll(group.getGuests()); }  
  
    // public void removeGroupFromRoom() { }
```

- A Class that inherits from the previous class

The Bedroom class is the subclass of the Room class. Apart from name, the information from the fields comes from the enum of the bedroom type.

```

public class Bedroom extends Room {
    private BedroomType type;

    public Bedroom(String name, BedroomType type) {
        super(name, type.getCapacity(), type.getPrice(), type.getExclusive());
        this.type = type;
    }
}

public enum BedroomType {
    SINGLE(50, 1, true),
    DOUBLE(80, 2, true),
    FAMILY(90, 4, true);

    private final double price;
    private final int capacity;
    private final boolean exclusive;

    private BedroomType(double price, int capacity, boolean exclusive) {
        this.price = price;
        this.capacity = capacity;
        this.exclusive = exclusive;
    }

    public double getPrice() { return this.price; }
    public int getCapacity() { return this.capacity; }
    public boolean getExclusive() { return this.exclusive; }
}

```

- An Object in the inherited class

```

public void before(){
    bedroom1 = new Bedroom( name: "Room 1", BedroomType.DOUBLE);
}

```

- A Method that uses the information inherited from another class.

The getCapacity() method is inherited from the room class and so a bedroom object can use it.

```

6 public class BedroomTest {
7
8     Bedroom bedroom1;
9
10
11     @Before
12     public void before(){
13         bedroom1 = new Bedroom( name: "Room 1", BedroomType.DOUBLE);
14     }
15
16
17     @Test
18     public void canGetCapacity(){
19         assertEquals( expected: 2, bedroom1.getCapacity());
20     }
21
22 }

```

BedroomTest

1 test passed - 3ms

ms /Library/Java/JavaVirtualMachines/jdk-9.jdk/Contents/Home/bin/java ...

ms Process finished with exit code 0

### IT 3

Function that searches data

```
def self.shop_transactions(id)
  sql = 'SELECT transactions.* FROM transactions INNER JOIN shops ON shops.id =
  transactions.shop_id WHERE shops.id = $1 ORDER BY date_of_transaction DESC'
  values = [id]
  transactions = SqlRunner.run(sql, values)
  result = transactions.map {|transaction| Transaction.new(transaction)}
end
```

The result of the function running

This function finds all the transactions for a particular shop.

Initially my table of transactions is:

Date	Amount	Shop	Category	Comment	Edit transaction	Delete transaction
2017-12-10	£149.57	British gas	household bills	Expensive as cold autumn	<button>Edit</button>	<button>Delete</button>
2017-12-07	£13.00	Co-op	Entertainment		<button>Edit</button>	<button>Delete</button>
2017-12-02	£25.00	Nandos	Entertainment		<button>Edit</button>	<button>Delete</button>
2017-12-01	£50.00	Primark	Clothing		<button>Edit</button>	<button>Delete</button>
2017-12-01	£0.50	Sainsbury	food		<button>Edit</button>	<button>Delete</button>
2017-12-01	£20.49	Marks and Spencer	Christmas presents	present for mum	<button>Edit</button>	<button>Delete</button>
2017-12-01	£19.99	Lothian buses	transport		<button>Edit</button>	<button>Delete</button>
2017-11-30	£160.00	Co-op	household bills		<button>Edit</button>	<button>Delete</button>
2017-11-29	£1090.99	Lothian buses	transport		<button>Edit</button>	<button>Delete</button>

After the function has been run the new table is:

Date	Amount	Shop	Category	Comment
2017-12-07	£13.00	Co-op	Entertainment	
2017-11-30	£160.00	Co-op	household bills	

### IT 4

Functions that sort data

The result of the function running

The function orders the transactions by date with the latest in time appearing at the top of the list and limits the table to only show five results

```
def self.last_five()
  sql = "SELECT * FROM transactions ORDER BY date_of_transaction DESC LIMIT 5"
  transactions = SqlRunner.run(sql)
  result = transactions.map {|transaction| Transaction.new(transaction)}
end
```

The result of running the function shows five results with the latest at the top:

Date	Amount	Shop	Category	Comment
2017-12-10	£149.57	British gas	household bills	Expensive as cold autumn
2017-12-07	£13.00	Co-op	Entertainment	
2017-12-02	£25.00	Nandos	Entertainment	
2017-12-01	£19.99	Lothian buses	transport	
2017-12-01	£50.00	Primark	Clothing	

## IT 5

An array in a function

A function that uses the array

The result of the function running

```
array.rb — ~/Documents/codeclan/PDA evidence
1  dessert_array = ["ice cream", "chocolate cake", "tiramisu" ]
2
3  def i_like_desserts_function(array)
4
5      for dessert in array
6          if dessert.include?("chocolate")
7              puts "I LOVE #{dessert}"
8          else
9              puts "I like #{dessert}!"
10         end
11     end
12
13 end
14
15 i_like_desserts_function(dessert_array)
16
```

```
PDA_evidence — benjaminbc
→ PDA_evidence ruby array.rb
I like ice cream!
I LOVE chocolate cake
I like tiramisu!
→ PDA_evidence
```

## IT 6

A hash in a program

A function that uses a hash

The result of the function running

```
hashes.rb — ~/Documents/codeclan/PDA_evidence
PDA_evidence — benjaminbowen

countries_hash = [ {name: 'Norway', capital: 'Oslo', language: 'Norwegian' },
  {name: 'Spain', capital: 'Madrid', language: 'Spanish'},
  {name: 'France', capital: 'Paris', language: 'French'},
  {name: 'Argentina', capital: 'Buenos Aires', language: 'Spanish'},
  {name: 'Colombia', capital: 'Bogota', language: 'Spanish'}]

def find_spanish_speaking(hash)
  spanish_list = []
  for country in hash
    spanish_list << country[:name] if country[:language] == "Spanish"
  end
  return spanish_list
end

p find_spanish_speaking(countries_hash)

→ PDA_evidence ruby hashes.rb
["Spain", "Argentina", "Colombia"]
→ PDA_evidence
```

IT 7

Demonstrate the use of Polymorphism in a program.

Both the Cat and Dog classes implement the IRun interface:

```
public class Cat extends Animal implements IRun {
    public String run(){
        return "I am as faster as a cheetah - look at me run!";
    }
    public String roar(){
        return "Meow!";
    }
}

public interface IRun {
    public String run();
}

public class Dog extends Animal implements IRun {
    public String run(){
        return "I am running like a wolf";
    }
    public String roar(){
        return "woof!";
    }
}
```

A PetWalker has an ArrayList of IRuns:

```
2 public class PetWalker {
3
4     private String name;
5     private ArrayList<IRun> pets;
6
7     public PetWalker(String name){
8         this.name = name;
9         this.pets = new ArrayList<>();
10    }
11
12    public String getName() {
13        return name;
14    }
15
16    public ArrayList<IRun> getPets() {
17        return pets;
18    }
19
20    public void addPet(IRun pet){
21        this.pets.add(pet);
22    }
23
24    public void removePet(IRun pet){
25        this.pets.remove(pet);
26    }
27
28    public int getNumberOfPets(){
29        return this.pets.size();
30    }
31
32 }
33
34
```

As both Cat and Dog implement the IRun interface, both can be added to the pets ArrayList of the PetWalker:

```
7
8   Cat cat;
9   Dog dog;
10  PetWalker petWalker;
11
12  @Before
13  public void before(){
14      cat = new Cat();
15      dog = new Dog();
16      petWalker = new PetWalker( name: "Steve");
17  }
18
19  @Test
20  public void canAddPets(){
21      petWalker.addPet(cat);
22      petWalker.addPet(dog);
23      assertEquals( expected: 2, petWalker.getNumberOfPets());
24  }
25
26  }
27
PetWalkerTest
>> 1 test passed - 1ms
ns /Library/Java/JavaVirtualMachines/jdk-9.jdk/Contents/Home/bin/java ...
ns Process finished with exit code 0
```