The University of the West Indies, St. Augustine
COMP 3607 Object Oriented Programming II
2020/2021 Semester 1
Lab Tutorial - Week 8

This tutorial focuses on code smells and code refactoring.

Learning Objectives:

- Identify code smells in code snippets
- Describe problems with the code
- Refactor code to eliminate code smells

For each question, (1) Identify the specific code smell (2) Identify the main category that the smell belongs to and (3) suggest how the code can be refactored:

1. A program class that calculates Simple Interest:

```
public class SimpleInterest{
   private simpleInterest; //The simple interest the person has to pay
   private rate; //The rate in which the simple interest is calculated
   private principal; //The amount
   private time; // The time taken to pay the loan
}
```

2. A flight class in which packages can be added:

```
public class Flight{
    private String flightNumber;
    private String destination;
    private Package [] packages;
    private static int numPackages=0;
    private String status;

    public Flight(String flightNumber,String destination){
        this.flightNumber=flightNumber;
        this.destination=destination;
        packages=new Package[10];
        status="On Time";
    }

    public String getPackageDetails(int ID,String owner){
        for(int i=0;i<numPackages;i++){
            if(packages[i].getPackageID() == ID){
               String str=" Flight Number: " + flightNumber + " Destination " + destination + " ";
               return str;
            }
            else{
                if(packages[i].getOwner().equals(owner)){
                   String str=" Flight Number: " + flightNumber + " Destination " + destination + " ";
                    return str;
                }
            }

       }
       return null;
}
```

3. Imagine having this class for customers:

```
public class Customer {
 private String name;
 private String address;
 public Customer(String name, String address){
     this.name = name;
     this.address = address;
 }
 public String getName(){ return name; }
 public String getAddress(){ return address;}
}
```

Our system is being developed to hold the basic information for record keeping purposes. Sometime in the future we would like to generate an email to send to each customer to verify that their information is correct, however those specifications have yet to be defined. We append the class in the following way to anticipate this change:

```
public class Customer {
    private String name;
    private String address;
    private eContent;
    public Customer(String name, String address){
        this.name = name;
        this.address = address;
    }
    public String getName(){return name;}
    public String getAddress(){return address;}
    public String getEContent(){return eContent;}
    public void setEContent (String content){this.eContent = content ;}

    public  String emailContent(){
        String content = "Hello " + this.getName() + ", Is this address correct : " + this.getAddress() + "?";
        return content;
    }
}
```

4. In the below example, every method debit(), transfer(), and sendWarningMessage() has one validation statement that specifies that an account balance should be more than 500.

```
ie  if(amount <= 500) {
           throw new Exception("Minimum balance should be over 500");
     }
```

If the developer decides to change that value from 500, or he decides to add further conditions for validation, he must also locate and make modifications in each of these 3 methods.

```
public class Account {
       private String type;
       private String accountNumber;
       private int amount;

       public Account(String type,String accountNumber,int amount){
              this.amount=amount;
              this.type=type;
              this.accountNumber=accountNumber;
       }

       public void debit(int debit) throws Exception{
              if(amount <= 500)
                     throw new Exception("Minimum balance should be over 500");
              amount = amount-debit;
              System.out.println("Now amount is" + amount);
      }

       public void transfer(Account from,Account to,int creditAmount) throws Exception{
              if(from.amount <= 500)
                     throw new Exception("Minimum balance should be over 500");
              to.amount = amount+creditAmount;
       }

       public void sendWarningMessage(){
              if(amount <= 500)
                     System.out.println("amount should be over 500");
       }
}
```

5. Initially the method `getTotalPrice` is in the `Basket` class, but it only uses data belonging to the `Item` class:

```
class Item { .. }
class Basket {
    // ..
    float getTotalPrice(Item i) {
        float price = i.getPrice() + i.getTax();
        if (i.isOnSale())
            price = price - i.getSaleDiscount() * price;
        return price;
    }
}
```

6. The following example shows a parent class, the Employee class, being inherited by two other classes, FullTimeEmployee and PartTimeEmployee. As a rule of inheritance, both subclasses will inherit the name and hourlyRate attributes as well as the calcPay method. However, the problem occurs when the PartTimeEmployee class does not use the inherited calcPay method. Instead it overrides it and adds its own functionality.

```
public abstract class Employee {
	private String name;
	private double hourlyRate;

	public Employee (String name, double hourlyRate) {
		this.name = name;
		this.hourlyRate = hourlyRate;
	}

	public double calcPay (double normalHours, double overtimeHours) {
		return ((normalHours * hourlyRate) + (overtimeHours * 1.5 * hourlyRate));
	}
}

public class FullTimeEmployee extends Employee {
	public FullTimeEmployee (String name, double hourlyRate) {
		super(name, hourlyRate);
	}
}

public class PartTimeEmployee extends Employee {
	public FullTimeEmployee (String name, double hourlyRate) {
		super(name, hourlyRate);
	}
	@override
	public double calcPay (double normalHours, double overtimeHours) {
		return ((normalHours * hourlyRate) + (overtimeHours * 2.0 * hourlyRate)); // NOTE: 2.0 vs 1.5
	}
}
```