Assignment 2

Rasika Mohod
rmohod@gmu.edu
G01044774

Introduction to Software Testing (Edition 2): Book by Jeff Offutt and Paul Amman
Exercises Chapter 1, Number 7; Page 14.

Original code:

Vehicle.java

```
public class Vehicle implements Cloneable{

	protected int x;

	public Vehicle(int y)
	{
		x=y;
	}

	public Object clone()
	{
		Object result = new Vehicle(this.x);
		//Location "A"
		return result;
	}
}
```

Truck.java

```
public class Truck extends Vehicle {

	private int y;

	public Truck(int z)
	{
		super(z);
		y=z;
	}

	public Object clone()
	{
		Object result = super.clone();
		//Location "B"
		((Truck) result).y = this.y;
		return result;	}

}
```

CloneTest.java

```
public class Client {

	public static void main(String args[])
	{
		Truck suv = new Truck(4);
		Truck co = suv.clone();

		if(suv.x==co.x)
		{
			System.out.println("Output 1: True"+ co.x);
		}

		if(suv.getClass()==co.getClass())
		{
			System.out.println("Output 2: True "+ co.getClass());
		}
	}
}
```

a) Fault: Location "A"

According to Bloch, Item 11,

The provision that x.clone().getClass() should generally be identical to x.getClass() , however, is too weak. In practice, programmers assume that if they extend a class and invoke super.clone from the subclass, the returned object will be an instance of the subclass. The only way a superclass can provide this functionality is to return an object obtained by calling super.clone. If a clone method returns an object created by a constructor, it will have the wrong class. Therefore, if you override the clone method in a nonfinal class, you should return an object obtained by invoking super.clone. If all of a class's superclasses obey this rule, then invoking super.clone will eventually invoke Object 's clone method, creating an instance of the right class.

Thus, the fault is at Location "A" in original code, as the clone method is calling constructor of the class to create a new Vehicle object. This fault does not allocate/return correct class type to the subclass'c clone method.

The **proposed modification to code** is:

```
	public Object clone() throws CloneNotSupportedException
	{
		try{
			Object result = super.clone();
			((Vehicle)result).x = this.x;
			//Location "A"
			return result;
		}
		catch (CloneNotSupportedException e)
		{
			throw e;
		}
```

```
    }
```

Error observed:

```
Exception in thread "main" java.lang.ClassCastException: Vehicle cannot be
cast to Truck
    at Truck.clone(Truck.java:16)
    at CloneTest.main(CloneTest.java:7)
```

b)

Test case that does not execute the fault is not possible, as all uses of clone executes the fault (super class's clone method calls constructor to create object). Anytime subclass tries to clone the object, the super class's clone method is invoked through sub class's clone method (super.clone) and thus the fault is executed (super class's clone method calls constructor to create object).

c)

Test case that executes the fault but does not result in error is not possible as every time clone is attempted, the super class's clone method returns incorrect object type and thus result in ClassCastException in sub class's clone method.

d)

Test case that results in an error but not a failure:

```
Vehicle suv = new Vehicle(4);
Vehicle co = (Vehicle) suv.clone();
```

Expected:

```
Output 1: True , value of co.x = 4
Output 2: True , value of co.getClass() = class Vehicle
```

Actual:

```
Output 1: True , value of co.x = 4
Output 2: True , value of co.getClass() = class Vehicle
```

However, such test case is not possible if cloning is performed on sub class's object.

e) Error State:

```
suv = Truck
```

- x = 4
- y = 4

```
result = Vehicle
```

- x = 4

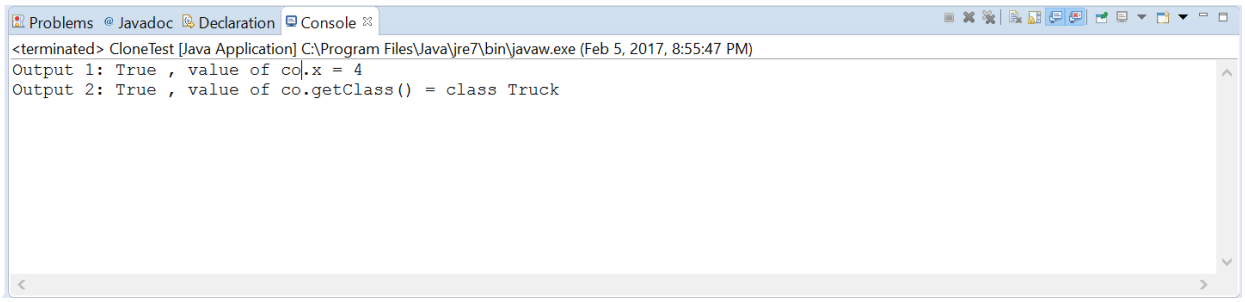
```
PC: at the start of Constructor of Vehicle class
```

f) Expected output generated after code repair:

```
Output 1: True , value of co.x = 4
```

```
Output 2: True , value of co.getClass() = class Truck
```

Screenshot for reference:



Repaired code:

Vehicle.java

```
public class Vehicle implements Cloneable{

	protected int x;

	public Vehicle(int y)
	{
		x=y;
	}

	public Object clone() throws CloneNotSupportedException
	{
		try{
			Object result = super.clone();
			((Vehicle)result).x = this.x;
			//Location "A"
			return result;
		}
		catch (CloneNotSupportedException e)
		{
			throw e;
		}
	}
}
```

Truck.java

```
public class Truck extends Vehicle {

	private int y;

	public Truck(int z)
	{
		super(z);
		y=z;
	}

	public Truck clone() throws CloneNotSupportedException
```

```
	{
		try{
			Truck result = (Truck) super.clone();
			result.y = this.y;
			return result;
		}
		catch(CloneNotSupportedException e)
		{
			throw e;
		}
	}
}
```

CloneTest.java

```
public class CloneTest {

	public static void main(String args[]) throws
	CloneNotSupportedException
	{
		Truck suv = new Truck(4);
		Truck co = suv.clone();

		if(suv.x==co.x)
		{
			System.out.println("Output 1: True , value of co.x = "+
			co.x);
		}

		if(suv.getClass()==co.getClass())
		{
			System.out.println("Output 2: True , value of co.getClass()
			= "+ co.getClass());
		}

	}
}
```