BSCCS2005: Graded Assignment with Solutions
Week 8

1. Consider the code given below. [MCQ:2 points]

```
public class Point{
    private int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public void setX(int x) {
        this.x = x;
    }
    public void setY(int y) {
        this.y = y;
    }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
    public Object clone() throws CloneNotSupportedException{
        return super.clone();
    }
}

public class FClass{
    public static void main(String[] args) {
        try {
            Point p1 = new Point(10, 20);
            Point p2 = p1;
            Point p3 = (Point)p1.clone();
            p1.setX(100);
            p1.setY(200);
            System.out.println(p1 + " , " + p2 + ", " + p3);
        }
        catch(CloneNotSupportedException e) {
            System.out.println("clone() not supported");
        }
    }
}
```

What will the output be?

- ◯ `(100, 200), (100, 200), (100, 200)`
- ◯ `(100, 200), (100, 200), (10, 20)`
- ◯ `(100, 200), (10, 20), (10, 20)`
- [✓ `clone() not supported`]

Solution: Since class `Point` does not implement `Cloneable`, an attempt to call `clone()` would generate `CloneNotSupportedException` exception. Thus, it prints `clone() not supported`.

2. Consider the code given below. [MCQ:2 points]

```
public class Product implements Cloneable{
    private String prodname;
    private double prodprice;
    public Product(String prodname, double prodprice) {
        this.prodname = prodname;
        this.prodprice = prodprice;
    }
    public Product(Product p) {
        this.prodname = p.prodname;
        this.prodprice = p.prodprice;
    }
    public void setProdname(String prodname) {
        this.prodname = prodname;
    }
    public void setProdprice(double prodprice) {
        this.prodprice = prodprice;
    }
    public String toString() {
        return prodname + " : " + prodprice;
    }
    protected Product clone() throws CloneNotSupportedException{
        return (Product)super.clone();
    }
}

public class FClass{
    public static void main(String[] args) {
        try {
            Product p1 = new Product("Pen", 100.0);
            Product p2 = new Product(p1);
            Product p3 = p1;
            Product p4 = p1.clone();
            p1.setProdname("Pencil");
            p1.setProdprice(30.0);
            System.out.print(p1 + ", " + p2 + ", " + p3 + ", " + p4);
        }
        catch(CloneNotSupportedException e) {
            System.out.println("clone() not supported");
        }
    }
}
```

What will the output be?

- ○ `Pencil : 30.0, Pencil : 30.0, Pencil : 30.0, Pencil : 30.0`
- ○ `Pencil : 30.0, Pen : 100.0, Pencil : 30.0, Pencil : 30.0`
- ✓ `Pencil : 30.0, Pen : 100.0, Pencil : 30.0, Pen : 100.0`
- ○ `clone() not supported`

Solution: Since `p2` allocates a new and copies the instance variables from `p1` (using copy constructor), the changes in `p1` is not reflected on `p2.`
Since, `p1` and `p3` refers to the same object, any change to `p1` would be reflected on `p3`.
However, `p4` creates a separate copy of the `p1` object. Thus, the changes in `p1` are not reflected on `p4.`

3. Consider the code given below. [MCQ:2 points]

```
public class FClass{
    public static void main(String[] args) {
        var a = 10;
        var b = "20";
        var c = a + b + 30;
        var d = a + 30 + b;
        System.out.println(c + ", " + d);
    }
}
```

What will the output be?

- ◯ `60, 60`
- ◯ `3030, 4020`
- ✓ `102030, 4020`
- ◯ `102030, 103020`

Solution: `a` has inferred type `int`.
`b` has inferred type `String`.
Thus, `a + b + 30 = "1020" + 30 = "102030"`,
and `a + 30 + c = 40 + "20" = "4020"`.

4. Consider the code given below. [MCQ:2 points]

```
public class Employee{
    public Employee(){}
    public String toString(){
        return "from Employee";
    }
}
public class Manager extends Employee{
    public Manager(){}
    public String toString(){
        return "from Manager";
    }
}
public class FClass{
    public static void main(String[] args) {
        Employee e = new Manager();
        var o1 = e;
        var o2 = new Employee();
        var o3 = new Manager();
        System.out.println(o1);
        System.out.println(o2);
        System.out.println(o3);
    }
}
```

What will the output be?

- ◯
```
from Employee
from Employee
from Manager
```
- [✓]
```
from Manager
from Employee
from Manager
```
- ◯
```
from Manager
from Manager
from Manager
```
- ◯
```
from Employee
from Employee
from Employee
```

Solution: `o1` has inferred type `Manager`.
`o2` has inferred type `Employee`.
`o3` has inferred type `Manager`.

5. Consider the code given below. [MCQ:2 points]

```
public class FClass{
    public static void main(String[] args) {
        var a = 100;
        a = 10.5;    //LINE 1
        var b = 5;
        var c = a / b;
        System.out.println(c);
    }
}
```

Choose the correct option regarding the code.

- ◯ It generates output: `2.1`
- ◯ It generates output: `2`
- ◯ It generates output: `20`
- ✓ It generates a compiler error at `LINE 1` due to incompatible types `int` and `double`.

Solution: For the statement `var a = 100;`, the type of `a` is inferred from the initial value. So, `a` is a `int`. Thus, the compiler does not allow `a = 10.5;`.

6. The `merge` method of `Map` has three arguments - key, value and reference to a function accepting two arguments - and merges the old value with the new value for a given key. Consider the code given below. [MCQ:2 points]

```
import java.util.*;
public class FClass{
    public static void main(String[] args){
        Map<String, Integer> order1 = new TreeMap<String, Integer>();
        order1.put("Pen", 3);
        order1.put("Pencil", 10);
        order1.put("Notebook", 4);
        order1.put("Paper", 50);
        Map<String, Integer> order2 = new TreeMap<String, Integer>();
        order2.put("Pencil", 20);
        order2.put("Eraser", 5);
        order2.put("Paper", 10);
        order2.put("Pen", 7);
        Map<String, Integer> totalSell = new TreeMap<String, Integer>();

        for(Map.Entry<String, Integer> e : order1.entrySet())
            totalSell.put(e.getKey(), e.getValue());

        for(Map.Entry<String, Integer> e : order2.entrySet())
            totalSell.merge(e.getKey(), e.getValue(), (x, y) -> y + x);

        System.out.println(totalSell);
    }
}
```

Choose the correct option regarding the code.

- [✓ It generates output: {`Eraser=5, Notebook=4, Paper=60, Pen=10, Pencil=30`}]
- ○ It generates output: {`Eraser=5, Notebook=4, Paper=10, Pen=7, Pencil=20`}
- ○ It generates output: {`Eraser=5, Notebook=4, Paper=50, Pen=3, Pencil=10`}
- ○ It generates runtime exception: `NullPointerException`

Solution: For each entry in `order2`, The statement `totalSell.merge(...)`, finds out if the key is already present in the `totalSell`.If the key does exists in `totalSell`, it would be added to `totalSell` alng with corresponding value. Otherwise, it gets the old value corresponding to the key, add it with the new value, and update `totalSell`.

7. Consider the code given below. [MSQ:2 points]

```
import java.util.stream.*;
import java.util.*;
public class Product{
    private String name;
    private double price;
    public Product(String n, double p){
        name = n;
        price = p;
    }
    public double getPrice(){
        return price;
    }
    public String toString(){
        return name + " : " + price;
    }
}
public class FClass{
    public static void main(String[] args){
        var pList = new ArrayList<Product>();
        pList.add(new Product("Pen", 10.0));
        pList.add(new Product("Pencil", 5.0));
        pList.add(new Product("Notebook", 40.0));
        pList.add(new Product("Eraser", 8.0));

        var outputList = ____________________;    //LINE 1
        outputList.forEach(n -> System.out.println(n));
    }
}
```

Identiy the appropriate option(s) to fill in the blank at `LINE 1` such that the output of the program is:

```
Pen : 10.0
Notebook : 40.0
```

- ✓ `pList.stream().filter(x -> x.getPrice() >= 10)`
- ◯ `pList.stream().filter(x -> x >= 10)`
- ✓ `pList.stream().filter((Product x) -> x.getPrice() >= 10)`
- ◯ `pList.stream().takeWhile(x -> x.getPrice() >= 10)`

Solution: The given program extracts the `Product` objects from the `pList` that have `price >= 10`. In order to access `price`, the accessor function `getPrice` is required to be invoked in `filter`.

8. Consider the code given below. [MCQ:2 points]

```
import java.util.stream.*;
import java.util.*;

public class FClass{
    public static void main(String[] args){
        int m = 15;
        Stream.iterate(1, n -> n + 1)
            .limit(m)
            .filter(n -> m % n == 0)
            .forEach(n -> System.out.print(n + " "));
    }
}
```

Identify the appropriate option for the above code.

- ◯ It produces no output.
- √ It produces output as `1 3 5 15`
- ◯ It produces output as `1 2 3 4 5 6 7 8 9 10 11 12 13 14 15`
- ◯ It generates compiler error due to invalid pipeline

Solution: `iterate(1, n -> n + 1)` generate a stream $123\cdots$.
`limit(m)` limit the stream at $m = 15$.
`filter(n -> m % n == 0)` filters the values which divides m, i.e. 13515. `forEach(n -> System.out.print(n + " ")` prints each element.

9. Consider the Java code given below, and answer the question that follows.

[MCQ:2pts]

```java
import java.util.stream.Stream;
import java.util.*;

public class StreamEx{
    int j=0;
    public static void main(String []args){
        ArrayList<Integer> list = new ArrayList<Integer>();

        for(int i = 1; i< 10; i++){
            list.add(i);
        }
        //CODE BLOCK 1
    }
}
```

From among the options, what should be filled in `CODE BLOCK 1` so that the code prints the even numbers between 1 and 9?

- [✓]
```java
Stream<Integer> stream = list.stream().filter(j -> j%2 == 0);
stream.forEach(s -> System.out.println(s));
```
- ○
```java
Stream<Integer> stream = list.stream().filter(j -> j%2 = 0);
stream.forEach(s -> System.out.println(s));
```
- [✓]
```java
Stream<Integer> stream = list.stream();
stream = stream.filter(j -> j%2 == 0);
stream.forEach(s -> System.out.println(s));
```
- ○
```java
Stream<Integer> stream = list.stream();
List<Integer> newList = (List)stream.filter(j -> j%2 == 0);
newList.forEach(s -> System.out.println(s));
```

Solution: Option 1 is correct.
In Option 2, the filtering condition must return a boolean value, here it is an assignment.
Option 3 is only a split form of Option 1, and hence is correct. In Option 4, the output of filter cannot be directly assigned to a List object (typecasting is not possible from Stream to List).

10. What is the likely outcome of the following code? [MCQ:2pts]

```
import java.util.stream.Stream;
public class StreamRandom {
    public static void main(String[] args) {
        Stream random = Stream.generate(Math::random)
                              .map(i -> Math.round(i * 100))
                              .filter(j -> j > 50).limit(5);
        random.forEach(s -> System.out.println(s));
    }
}
```

[✓ 92
71
98
96
52]

○ 78
39
49
29
54

○ 53
94
86
75
82
74

○ 0.14161383465857424
0.9951624577496674
0.1500299618014398
0.8523339358885837
0.20299930897974205

○ 6200
7400
9100
5100
5800

Solution: `Math::round` generates random numbers between 0.0 and 1.0. `Math.round(i * 100)` multiplies it by 100 and rounds it to an integer. `filter(j`

`-> j > 50)` filters in only those numbers that are greater than 50. `limit(5)` limits the number of numbers generated to 5. Option 1 is the only option that satisfies all the conditions.