BSCCS2005: Practice Assignment with Solutions
Week 6

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

```
import java.util.*;
public class FClass{
    public static void main(String[] args) {
        ArrayList<String> empList = new ArrayList<String>();
        empList.add("raj");
        empList.add("akash");
        empList.add("biraj");
        empList.add("vinay");
        ______________________________    //LINE 1
        while(iter.hasNext()) {
            System.out.print(iter.next() + " ");
        }
        System.out.println();
        while(iter.hasPrevious()) {
            System.out.print(iter.previous()  + " ");
        }
    }
}
```

Identify the appropriate option to fill in the blank at `LINE 1`, such that the output of the above code is

```
raj akash biraj vinay
vinay biraj akash raj
```

- ○ `Iterator<String> iter = empList.iterator();`
- √ `ListIterator<String> iter = empList.listIterator();`
- ○ `Iterator<String> iter = empList.listIterator();`
- ○ `ListIterator<String> iter = empList.iterator();`

Solution: Since it requires the capabilities `hasPrevious()` and `previous()`, it need to use an iterator of object of type `ListIterator`, which can be achived using `listIterator()` function.

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

```
import java.util.*;
public class Process{
    private int pid;
    public Process(int pid) {
        this.pid = pid;
    }
    public int getPID() {
        return pid;
    }
}
public class FClass {
    public static void main(String[] args){
        Queue<Process> pq = new LinkedList<Process>();
        for (int i = 0; i < 5; i++)
            pq.add(new Process(i + 1000));

        while(!pq.isEmpty()) {
            Process curPorc = _____________;    //LINE 1
            System.out.print(curPorc.getPID() + " -> ");
        }
    }
}
```

Identify the appropriate option(s) to fill in the blank at `LINE 1`, such that the output is:
`1000 -> 1001 -> 1002 -> 1003 -> 1004 ->`

- ○ `pq.peek()`
- ✓ `pq.remove()`
- ✓ `pq.poll()`
- ○ `pq.element()`

Solution: Since the `while` loop checks for if the queue is empty, within the loop the statement at `LINE` must return the element from the front of the queue and remove it from the queue. Thus, option-2 and option-3 are correct.

3. Consider the following code. [MCQ:2 points]

```
import java.util.*;
public class Shop{
  private String name;
  private int nsold; // number of items sold
  public Shop(String s, int ns){
    this.name = s;
    this.nsold = ns;
  }
  public String getName(){
    return name;
  }
  public int getItemSold(){
    return nsold;
  }
}

public class Test {
  public static void main(String[] args) {
    Shop s1 = new Shop("BigBazaar", 20);
    Shop s2 = new Shop("BigBazaar", 20);
    Shop s3 = new Shop("SV stores", 12);
    Shop s4 = new Shop("SunGeneral", 10);
    HashMap<String, Integer> m = new HashMap<String, Integer>();
    m.put(s1.getName(), m.getOrDefault(s1.getName(),0)+s1.getItemSold());
    m.put(s2.getName(), m.getOrDefault(s2.getName(),0)+s2.getItemSold());
    m.put(s3.getName(), m.getOrDefault(s3.getName(),0)+s3.getItemSold());
    m.put(s4.getName(), m.getOrDefault(s4.getName(),0)+s4.getItemSold());
    String shop = "";
    int sold = 0;
    -----------SEGMENT 1----------------
  }
}
```

At the end of execution, the variable `shop` should store the name of shop, which has sold maximum number of items, and the variable `sold` should store the total number of items sold by that shop. (In the case of the given code, `shop` should store BigBazaar and `sold` should store 40).

Identify the appropriate option to fill in the blank at `SEGMENT 1`.

◯
```
for (HashMap.Entry<String, Integer> entry : m.entrySet()){
    if(entry.getValue()> shop) {
        shop = entry.getKey();
        sold = entry.getValue();
    }
}
```

○
```
    for (HashMap.Entry<String, Integer> entry : m.entrySet()){
        if(entry.getKey()> sold) {
            shop = entry.getKey();
            sold = entry.getValue();
        }
    }
```

○
```
    for (HashMap.Entry<Integer, Integer> entry : m.entrySet()){
        if(entry.getValue()> sold) {
            shop = entry.getKey();
            sold = entry.getValue();
        }
    }
```

√
```
    for (HashMap.Entry<String, Integer> entry : m.entrySet()){
        if(entry.getValue()> sold) {
            shop = entry.getKey();
            sold = entry.getValue();
        }
    }
```

Solution: `HashMap m` maps the name of the shop to total number of items sold by that shop. `sold` value must be updated only if the entry.getValue() $>$ sold.

4. Consider the following code. [MCQ:2 points]

```
import java.util.*;
public class Student{
  private String name;
  private int maths, physics, chemistry;
  Student(String s, int m, int p, int c){
    this.name = s;
    this.maths = m;
    this.physics = p;
    this.chemistry = c;
  }
  public String getName(){
    return name;
  }
  public int getMaths(){
    return maths;
  }
  public int getPhysics(){
    return physics;
  }
}
public class Test {
  public static void main(String[] args) {
    Student s1 = new Student("Ravi", 90, 55, 50);
    Student s2 = new Student("Ram" , 72, 80, 55);
    Student s3 = new Student("Ramu" , 50, 80, 55);
    ArrayList<Student> l1 = new<Student> ArrayList();
    ArrayList<Student> l2 = new<Student> ArrayList();
    l1.add(s1);
    l1.add(s2);
    l1.add(s3);
    for(Student s : l1){
      if(s.getMaths() > 80 && s.getPhysics() < 60){
        l2.add(s);
      }
    }
  }
}
```

Choose the correct option regarding the code.

- ◯ `l1` has 3 elements in it and `l2` has 3 elements in it.
- ◯ Compilation error because Student objects cannot be inserted in ArrayList

◯ Compilation error because > operator is not defined for Student type

[✓ `l1` has 3 elements in it and `l2` has 1 element in it]

Solution: `ArrayList l1` has 3 elements. elements from `ArrayList l1` are added to `ArrayList l2` if the elements of `ArrayList l1` has maths marks > 80 and physics marks < 60.

5. Consider the following code. [MCQ:2points]

```
import java.util.*;
public class Test{
  public static void main(String args[]) {
    LinkedList<String> obj = new LinkedList<String>();
    obj.add("A");
    obj.add("C");
    obj.add(0, "D");
    obj.add("B");
    Collections.sort(obj);
    System.out.println(obj);
  }
}
```

What will the output be?

- √ [A, B, C, D]
- ◯ [D, A, C, B]
- ◯ [A, C, D, B]
- ◯ [A, D, B, C]

Solution: `Collections.sort(obj)` sorts the elements of `obj` in ascending order.

6. Consider the Java program given below and choose a possible outcome of executing it.

[MCQ : 2 points]

```
import java.util.*;
public class Example {
    public static void main(String[] args) {
        List<String> list1=new ArrayList<String>();
        list1.add("IITM");
        list1.add("Java");
        list1.add("Java");
        list1.add("Programming");
        Set<String> set1=new HashSet<String>(list1);//Line 1
        for (String string : set1) {
            System.out.println(string);
        }
    }
}
```

- ○ Compile time error at Line 1
- ✓ This code generates output:
```
Java
Programming
IITM
```
- ○ This code generates output:
```
Java
Java
Programming
IITM
```
- ○ This code generates output:
```
null
null
null
```

Solution: Here, we pass an `ArrayList` object in the `HashSet` constructor in Line 1. Duplicate elements are skipped while adding elements to the set.

7. Consider the Java code given below and predict the output for Lines 1, 2, 3 and 4.

[MCQ : 2 points]

```
import java.util.*;
public class  Example{
    public static void main(String[] args){
        ArrayList<Integer> list=new ArrayList<Integer>();
        list.add(100);
        list.add(200);
        list.add(300);
        System.out.println(list.indexOf(100));//Line 1
        System.out.println(list.get(1));//Line 2
        HashSet<Integer> set=new HashSet<Integer>(list);
        System.out.println(set.indexOf(100));//Line 3
        System.out.println(set.get(2));//Line 4
    }
}
```

- ○ Line 1 prints 0
  Line 2 prints 200
  Line 3 prints 0
  Line 4 prints 300
- ○ Line 1 prints 1
  Line 2 prints 100
  Line 3 prints 1
  Line 4 prints 200
- ○ Line 1 prints 0
  Line 2 prints 100
  Line 3 prints true
  Line 4 prints 300
- √ Compilation errors at Line 3 and 4.

Solution: A `List` interface is an ordered collection. All classes that implement the `List` interface will guarantee a sorted order while storing the elements. Hence, we can invoke `indexOf()` method and `get()` method on such classes.
A `Set` interface, on the other hand, is an unordered collection. The classes implementing the `Set` interface do not guarantee sorted order for the elements. Hence, it is meaningless to invoke `indexOf()` and `get()` methods on such classes. Thus, Lines 3 and 4 gives compilation errors.

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

```
import java.util.*;
public class  ArrayDequeExample{
    public static void main(String[] args){
        ArrayDeque<String> deque1=new ArrayDeque<String>();
        deque1.push("IIT");
        deque1.push("Madras");
        deque1.push("Java");
        deque1.push("Object");
        deque1.push("Oriented");
        deque1.push("Programming");
        deque1.push("Language");
        ArrayDeque<String> deque2=new ArrayDeque<String>(deque1);
        for (int i=0;i<6;i++) {
           deque1.pop();
        }
        for (int i=0;i<6;i++) {
           deque2.peek();
        }
    }
}
```

How many elements are present in `deque1` and `deque2` after executing this code?

- ◯ `deque1` has no elements.
  `deque2` has no elements.
- ✓ `deque1` has 1 element.
  `deque2` has 7 element.
- ◯ `deque1` has 1 element.
  `deque2` has 1 element.
- ◯ `deque1` has 7 elements.
  `deque2` has 7 elements.

Solution: 7 elements are present in both `deque1` and `deque2` before calling `pop()` and `peek()`.
`pop()` is called 6 times on `deque1`.
`peek()` is called 6 times on `deque2`.
`pop()` will return and remove the element from `deque1`, whereas `peek()` will return the value from `deque2` without removing it.

9. Consider the Java program given below, and choose correct the options.

[MCQ : 2 points]

```
import java.util.*;
public class MapEx{
    public static void main(String[] args) {
        Map<String,String> map1;
        map1=new HashMap<String,String>();       //Line 1
        map1.put("India","Delhi");
        map1.put("Srilanka","Colombo");
        map1.put("Australia","Sydney");
        System.out.println(map1);               //Line 2
        map1=new TreeMap<String,String>(map1);  //Line 3
        System.out.println(map1);               //Line 4
    }
}
```

- ○ `Lines 2 and 4` always produce the same output.
- ✓ `Lines 2 and 4` may produce different output.
- ○ Compilation error at `Line 1`.
- ○ Compilation error at `Line 3`.

Solution: Applied indirection to the program.
`map1` reference variable instantiated with `HashMap` at `Line 1`.
`map1` reference variable instantiated with `TreeMap` at `Line 3`.
`map1` prints different output at `Line 2 and 4`.
`HashMap` can not store the values in sorted order.
`TreeMap` can store the values in sorted order.

10. Consider the code given below. Choose the correct option regarding the given code.

[MCQ:2 points]

```
import java.util.*;
public class Test{
    public static void main (String[] args){
        Map<String, Integer> map = new LinkedHashMap();
        String[] str = {"E","A","B","D","C"};
        Integer[] arr = {5,3,1,2,4};
        for(int i=0;i<str.length;i++){
           map.put(str[i],arr[i]);
        }
        Set s=map.entrySet();
        Iterator itr=s.iterator();
        while(itr.hasNext()){
           Map.Entry m = (Map.Entry)itr.next();
           if(m.getKey().equals("B")){
              m.setValue(2);
           }
           System.out.println(m.getKey()+" => "+m.getValue());
        }
    }
}
```

○ This program generates output:
A => 3
B => 2
C => 4
D => 2
E => 5

[✓ This program generates output:
E => 5
A => 3
B => 2
D => 2
C => 4]

○ This program generates output:
B => 2
D => 2
A => 3
C => 4
E => 5

○ This program generates compile time error since trying to assign same values to key `B` and `D`

Solution: The LinkedHashMap maintains the order in which key-value pairs are inserted.