BSCCS2005: Mock Quiz 2
Weeks 1-8

1. Consider the following code.

```
import java.util.*;
public interface Employee{
    public abstract void showSalary();
}
public class Manager implements Employee{
    double salary;
    public Manager(double amt){
        salary = amt;
    }
    public void showSalary() {
        System.out.println("Manager salary: " + salary);
    }
}
public class HiringManager implements Employee{
    double salary;
    public HiringManager(double amt){
        salary = amt;
    }
    public void showSalary() {
        System.out.println("HiringManger salary: " + salary);
    }
}
public class Test{
    public static void main(String args[]) {
        ____________________________ // LINE 1
        emp.add(new Manager(55000.0));
        emp.add(new HiringManager(40000.0));
        for(Employee e : emp) {
            e.showSalary();
        }
    }
}
```

If the code given above produces the output:

```
Manager salary:  55000.0
HiringManager salary:  40000.0
```

What should be the correct choice for LINE 1?

- ○ `ArrayList<Manager> emp = new ArrayList<Manager>();`
- ○ `ArrayList<HiringManager> emp = new ArrayList<HiringManager>();`
- [✓ `ArrayList<Employee> emp = new ArrayList<Employee>();`]
- ○ `ArrayList<Object> emp = new ArrayList<Object>();`

Solution:
Interface `Employee` is implemented by both the classes `Manager` and `HiringManager`. In order to call the respective version of the overridden method `showSalary()`, the `ArrayList` should store `Employee` type reference.

2. Consider the following code.

```
import java.util.*;
public class Vehicle implements Comparable<Vehicle>{
  int wheels;
  public Vehicle(int n){
    wheels = n;
  }
  public int compareTo(Vehicle v) {
        if(wheels < v.wheels)
            return 1;
        else if (wheels > v.wheels)
            return -1;
        else return 0;
  }
  public String toString(){
    return "" + wheels;
  }
}
public class Test{
    public static void main(String args[]) {
        Set<Vehicle> veh = new TreeSet<>();
        veh.add(new Vehicle(4));
        veh.add(new Vehicle(2));
        veh.add(new Vehicle(6));
        veh.add(new Vehicle(1));
        for(Vehicle v : veh) {
            System.out.print(v + " ");
        }
    }
}
```

What will the output be?

- ◯ 1 2 4 6
- ✓ 6 4 2 1
- ◯ 1 4 2 6
- ◯ This program does not compile

Solution: `TreeSet` stores the elements in sorted order. The class `Vehicle` implements interface `Comparable` and overrides the method `compareTo()` to sort the elements.

3. Consider the following code and choose the correct option.

```
public class Person{
  int age;
  String name;
  public Person(String n, int a){
    age = a;
    name = n;
  }
  public String toString(){
    return (name+ ":" + age );
  }
}
public class Employee extends Person implements Cloneable {
  public Employee(String n, int a){
    super(n, a);
  }
  public Employee clone()throws CloneNotSupportedException{
    return (Employee)super.clone();
  }
}

public class Test {
  public static void main(String[] args) {
    Employee[] e1 = {new Employee("Hari",30), new Employee("geeta",23)};
    Employee[] e2 = e1.clone();
    e2[1].name = "rani";
    System.out.println(e1[1] + ", " + e2[1]);
  }
}
```

- ◯ This program generates compilation error.
- ◯ This program generates the output:
  geeta:23, rani:23
- ◯ This program generates the output:
  geeta:23, geeta:23
- [✓ This program generates the output:
  rani:23, rani:23]

Solution: `e1, e2` are pointing to same memory, so `e1[1]` and `e2[1]` pointing to same location

4. Consider the following code and choose the correct option.

```
public interface ArrOperations<T extends Number>{
  public abstract void display(T[] arr);
}

public class Test {
  public static void main(String[] args) {
    Integer[] a = new Integer[3];
    a[0] = 12;
    a[1] = 13;
    a[2] = 14;
    ArrOperations<Integer> arr = (s) ->{
            for(int i = 0; i < s.length; i++){
                System.out.print(s[i] + 2 + " ");
            }
        };
    arr.display(a);
  }
}
```

- ◯ This program generates compilation error because of type mismatch.
- ◯ This program compiles successfully but does not produce output.
- ✓ This program generates the output:
  14 15 16
- ◯ This program generates compilation error because of incorrect usage of the lambda expression.

Solution: As the interface `ArrOperations` is a functional interface, lambda implementation is valid for abstract method display.

5. Consider the following code.

```
public class Department implements Cloneable{
  String name;
  public Department(String n){
    name = n;
  }
  public Department clone() throws CloneNotSupportedException{
    return (Department)super.clone();
  }
}
public class University implements Cloneable{
  Department dept;
  String Uname;
  public University(Department d, String n){
    dept = d;
    Uname = n;
  }
  public University clone() throws CloneNotSupportedException{
    ------------------------
      ** CODE SEGMENT **
    ------------------------
  }
  public String toString(){
    return Uname + ":" + dept.name;
  }
}
public class Test {
  public static void main(String[] args) {
    University u1 = new University(new Department("cse"), "xyz");
    try{
      University u2 = u1.clone();
      u2.dept.name = "ece";
      System.out.println(u1 + ","+ u2);
    }
    catch(Exception e){
      System.out.println(e);
    }
  }
}
```

Choose the correct option to fill in the `CODE BLOCK` so that the output is:
`xyz:cse,xyz:ece`

◯ `University u = super.clone();`

```
   return u;
```

○
```
University u = (University)super.clone();
return u;
```

○
```
University u = (University)super.clone();
u.dept = (University)dept.clone();
return u;
```

✓
```
University u = (University)super.clone();
u.dept = dept.clone();
return u;
```

Solution: As the class `University` has `Department dept` as instance variable, `dept` has to be cloned to perform deep copying.

6. Consider the following code.

```
import java.util.*;
public class Test {
    public static void main(String[] args) {
        Map<Set<String>, List<String>> m = new HashMap<>();
        ArrayList<String> list1 = new ArrayList<>();
        list1.add("30");
        list1.add("40");

        ArrayList<String> list2 = new ArrayList<String>();
        list2.add("35");
        list2.add("45");

        Set<String> set1 = new HashSet<>();
        set1.add("ravi");
        set1.add("ravi");
        m.put(set1,list1);
        m.put(set1,list2);
        System.out.println(m);
    }
}
```

What will the output be?

- [✓ `{[ravi] = [35, 45]}`]
- ○ `{[ravi] = [30, 40]}`
- ○ `{[ravi, ravi] = [30, 40]}`
- ○ `{[ravi, ravi] = [35, 45]}`

Solution: Duplicates are not allowed in `Set`. Update of values happens when we try to insert key-value pairs with duplicate keys.

7. Consider the following code.

```
import java.util.*;
import java.util.stream.*;
public class Test {
  public static void main(String[] args) {
     Optional<Double> d = Stream.generate(Math::random)
                         .limit(100)
                         .map((i) -> i + 10)
                         .max(Double::compareTo);
     System.out.println(d);
  }
}
```

Choose the correct option

- ◯ This program generates compilation error.
- ◯ This program generates the output:
  A random `Optional<Double>` between 0 and 1
- √ This program generates the output:
  A random `Optional<Double>` between 10 and 11
- ◯ This program generates the output:
  A stream of random `Optional<Double>` between 10 and 11

Solution: If the stream is not empty, `max()` returns an `Optional<Double>`.

8. Consider the following code.

```
import java.util.*;
class Col<T extends AbstractList>{
    T obj;
    Col(T obj){
        this.obj=obj;
    }
    public void display(){
        System.out.println(obj);
    }
}
public class Test{
    public static void main(String[] args) {
        //Line 1
        list.add("India");
        list.add("IIT");
        list.add("Madras");
        list.add("Java");
        Col col=new Col(list);
        col.display();
    }
}
```

Choose the correct option(s) for `Line 1` such that the program prints.

```
[India, IIT, Madras, Java]
```

- ✓ `ArrayList<String> list = new ArrayList<String>();`
- ✓ `LinkedList<String> list = new LinkedList<String>();`
- ○ `Set<String> list=new HashSet<String>();`
- ○ `Set<String> list=new TreeSet<String>();`

Solution: The statement `class Col<T extends AbstractList>` implies that `T` can be any subclass of the `AbstractList`.
Since `ArrayList` and `LinkedList` are the subclasses of the `AbstractList`, both the options give correct output.

9. Consider the following code.

```
class Bowler {
    String name;
    String type;
    public Bowler(String name, String type) {
        this.name = name;
        this.type = type;
    }
    public String toString() {
        return " [name=" + name + ", type=" + type + "]";
    }
}
class AllRounder extends Bowler {
    public AllRounder(String name, String type) {
        super(name, type);
    }
    public String toString() {
        return " [name=" + name + ", dept=" + type + "]";
    }
}
public class Test {
    public static <S extends T,T> void copy (S[] src,T[] tgt){
        int i,limit;
        limit = Math.min(src.length,tgt.length);
        for (i = 0; i < limit; i++){
            tgt[i] = src[i];
        }
    }
    public static void main(String[] args) {
        ------------------------------------
        ************CODE SEGMENT************
        ------------------------------------
        for (int i = 0; i < bowlers.length; i++) {
            System.out.println(bowlers[i]);
        }
    }
}
```

Choose the correct option to fill in the `CODE BLOCK` so that the output is:

```
[name=V.Iyer, dept=AllRounder]
[name=D.Hooda, dept=AllRounder]
```

◯
```
Bowler obj1 = new Bowler("V.Iyer", "AllRounder");
Bowler obj2 = new Bowler("D.Hooda", "AllRounder");
Bowler obj3 = new Bowler("R.Jadeja", "AllRounder");
Bowler bowler[] = {obj1,obj2,obj3};
AllRounder[] allRounders = new AllRounder[2];
Test3.copy(bowler, allRounders);
```

√
```
AllRounder obj1 = new AllRounder("V.Iyer", "AllRounder");
AllRounder obj2 = new AllRounder("D.Hooda", "AllRounder");
AllRounder obj3 = new AllRounder("R.Jadeja", "AllRounder");
AllRounder allRounders[] = {obj1,obj2,obj3};
Bowler[] bowlers = new Bowler[2];
Test3.copy(allRounders, bowlers);
```

◯
```
Bowler obj1 = new Bowler("V.Iyer", "AllRounder");
Bowler obj2 = new Bowler("D.Hooda", "AllRounder");
Bowler obj3 = new Bowler("R.Jadeja", "AllRounder");
Bowler bowler[] = {obj1,obj2,obj3};
AllRounder[] allRounders = new AllRounder[3];
Test3.copy(bowler, allRounders);
```

◯
```
AllRounder obj1 = new AllRounder("V.Iyer", "AllRounder");
AllRounder obj2 = new AllRounder("D.Hooda", "AllRounder");
AllRounder obj3 = new AllRounder("R.Jadeja", "AllRounder");
AllRounder allRounders[] = {obj1,obj2,obj3};
Bowler[] bowlers = new Bowler[3];
Test3.copy(allRounders, bowlers);
```

Solution: In the above program, while copying, the array's source should be a subtype of the target.
According to the statement above, options 1 and 3 are invalid.
According to the output, only two objects should be copied to the target array, hence option 2 should be valid.

10. Consider the following code.

```
import java.lang.reflect.*;
import java.util.ArrayList;
class Student{
    public static final String college = "IITM";     //Line 1
    private String name;
    public  int rollno;
    public Student() {}
    public Student(String name) {
        this.name = name;
    }
    private Student(int rollno) {   //Line 2
        this.rollno = rollno;
    }
    public String getName() {
        return name;
    }
    public int getRollno() {
        return rollno;
    }
}
public class Test{
    public static void main(String[] args) throws ClassNotFoundException {
        ArrayList<String> list = new ArrayList<String>();
        Class c = Class.forName("Student");
        Constructor[] consts = c.getConstructors();
        Field[] fields1 = c.getFields();
        for(Constructor i:consts)
            list.add(i.toString());
        for(Field i : fields1)
            list.add(i.toString());
        for(String i:list)
            System.out.println(i);
    }
}
```

Choose the correct option.

√ This program generates the output:

```
public Student()
public Student(java.lang.String)
public static final java.lang.String Student.college
public int Student.rollno
```

◯ This program generates compilation error at `Line 1`, because you cannot make `static` variable as `final`.

◯ This program generates compilation error at `Line 2`, because you cannot make constructor as `private`.

◯ This program generates the output:

```
public Student()
private Student(int)
public Student(java.lang.String)
public static final java.lang.String Student.college
private java.lang.String Student.name
public int Student.rollno
```

Solution:
`getConstructors()` is used to obtain the public constructors of a given class.
`getFields()` is used to obtain public instance variables of a given class.

11. Consider the following code.

```
public class Test {
    String msg = null;
    public void show(){
        try {
            try {
                System.out.println(10/0);
            }
            catch(ArithmeticException e) {
                msg = e.getMessage();     //Line 1
            }
            try {
                System.out.println("IITM".charAt(5));
            }
            catch(StringIndexOutOfBoundsException e) {
                msg = e.getMessage();     //Line 2
            }
        }
        finally {
            System.out.println(msg);
        }
    }
    public static void main(String[] args) {
        Test obj = new Test();
        obj.show();
    }
}
```

Choose the correct option.

- [✓] This program generates the output:
  `String index out of range:  5`
- ◯ Compilation errors at `Line 1` and `Line 2`.
- ◯ Program terminates abnormally due to unhandled exception(s).
- ◯ This program generates the output:
  `/ by zero`

Solution: In above program, in the first `try` block `ArithmeticException` will be raised, immediately control moves to its corresponding catch block and variable `msg` is initialized by the string associated with the `ArithmeticException`.

Now in the second `try` block `StringIndexOutOfBoundsException` will be raised, immediately control moves to its corresponding catch block and here the previous value in `msg` is overwritten by the string associated with the `StringIndexOutOfBoundsException`.

12. Consider the following code.

```
import java.util.*;
public class Test {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            int a = scanner.nextInt();
            int b = scanner.nextInt();
            int c = a/b;
        }
        catch(ArithmeticException e) {
            System.out.println("b value should not be zero");
        }
        catch(InputMismatchException e) {
            System.out.println("please give integer value as input");
        }
    }
}
```

If the input is given as:

```
10
0.0
```

Choose the correct option such that the output is:

- ○ This program generates the output:
  `b value should not be zero`
- [✓ This program generates the output:
  `please give integer value as input`]
- ○ Program terminates abnormally due to unhandled exception(s).
- ○ Compilation error.

Solution: Input given `a` value as 10 and `b` value as 0.0.
Here instead of an integer value, a double value 0.0 is given as input, hence InputMismatchException raised at runtime.

13. Consider the following code.

```
interface TV{
    abstract void features(String name);
}
class LedTv implements TV{
    public void features(String name) {
        System.out.println("Picture quality of "+name+" TV is good.");
    }
}
class SmartTv extends LedTv{
    public void features(String name) {
        super.features(name);
        System.out.println(name +" TV similar to LED TV with internet connection");
    }
}
public class Details {
    public <T extends TV> void getDetails(T obj,String name) { //Line 1
        obj.features(name);
    }
    public static void main(String[] args) {
        Details obj=new Details();
        obj.getDetails(new LedTv(),"Led");
        obj.getDetails(new SmartTv(),"Smart"); //Line 2
    }
}
```

Choose the correct option.

- ◯ This program generates compilation error at `Line 1`.
- ◯ This program generates compilation error at `Line 2`.
- ◯ This program generates the output:
  ```
  Picture quality of Led TV is good.
  Picture quality of Led TV is good.
  Smart TV is essentially an LED TV with internet connection
  ```
- √ This program generates the output:
  ```
  Picture quality of Led TV is good.
  Picture quality of Smart TV is good.
  Smart TV is essentially an LED TV with internet connection
  ```

Solution: `public <T extends TV> void getDetails(T obj,String name)`
The above statements tells that `T` can be any implemented class of interface `TV`
Hence, both `SmartTv` and `LedTv` we can pass to the `T`.

14. Consider the code given below.

```
import java.util.Scanner;
public class Test{
    public static String getSubString(int a, int b,String str){
        assert a < b: "Invalid index values";    //assert-1
        assert b < str.length(): "Invalid ending index";      //assert-2
        return str.substring(a,b);
    }
    public static void main(String[] args){
        Scanner scanner=new Scanner(System.in);
        int a = 10;
        int b = 5;
        String str="IITM";
        assert a > 0: "a should not be negative";      //assert-3
        assert b > 0: "b should not be negative";      //assert-4
        System.out.println(getSubString(a, b,str));
    }
}
```

Identify the first `assert` statement that throws the `AssertionError` when the program is executed as:
`java -ea Test`

[✓ `assert-1`]
◯ `assert-2`
◯ `assert-3`
◯ `assert-4`

Solution: The condition given for the assert statement `assert-1` is false, so it throws the `AssertionError`.

15. Consider the codes given below.

//File Name : Read.java

```
package com.read;
import java.util.Scanner;
public class Read {
    private int a,b;
    Scanner scanner=new Scanner(System.in);
    public void readValues() {
        a = scanner.nextInt();
        b = scanner.nextInt();
    }
}
```

//File Name : Add.java

```
public class Add {
    public void addition() {
        com.read.Read read=new com.read.Read();//Line 1
        read.readValues();//Line 2
        int result=read.a+read.b;//Line 3
        System.out.println(result);
    }
    public static void main(String[] args) {
        Add add = new Add();
        add.addition();
    }
}
```

Choose the correct option.

- ○ `Add.java` produces compilation error at `Line 1`, because of illegal access to the Read class.
- ○ `Add.java` produces compilation error at `Line 2`, because of illegal access to the readValues() method.
- [✓ `Add.java` produces compilation error at `Line 3`, because of illegal access to the variables `a` and `b`.]
- ○ All of the above.

Solution: Option 3 is correct.
You cannot access private variables directly outside the package.