BSCCS2005: Practice Assignment with Solutions
Week 5

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

```
public abstract class OutputDevice{
    public abstract void output();
}
public class Printer extends OutputDevice{
    public void output() {
         System.out.println("printer prints");
    }
}
public class Monitor extends OutputDevice{
    public void output() {
        System.out.println("monitor displays");
    }
}
public interface Iterable{
    public boolean has_next();
    public Object get_next();
}
public class OutputList implements Iterable{
    private final int max_limit = 2;
    private int indx;
    private Object[] oArr = {new Printer(), new Monitor()};
    public OutputList(){
        indx = -1;
    }
    public boolean has_next() {
        if (indx < max_limit - 1)
            return true;
        return false;
    }
    public Object get_next() {
        indx++;
        return oArr[indx];
    }
}
public class FClass{
    public static void main(String[] args) {
        OutputList list = new OutputList();
        while(list.has_next()) {
            //LINE1
        }
    }
}
```

Identify the appropriate option to fill in the blank at `LINE1` such that the output is

```
printer prints
monitor displays
```

- ◯ `(list.get_next()).output();`
- ◯ `((Monitor)list.get_next()).output();`
- [✓ `((OutputDevice)list.get_next()).output();`]
- ◯ `((OutputList)list.get_next()).output();`

Solution: `Object` type elements required to be type casted to `OutputDevice` type in order to access the `output` method.

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

[Bikash:MCQ:2 points]

```
public class Example<T>{
   T ob;
   Example(T x){
      this.ob=x;
   }
   public String show(){
      return ""+ob.getClass().getName();
   }
   public T get(){
      return ob;
   }
}
public class Test {
    public static void main(String[] args){
       Example<Number> n=new Example<Number>(100);
       Example<Double> e=new Example<Double>(10.5);
       n=e;
       System.out.print(n.show()+"\n"+n.get());
    }
}
```

- [✓ This program generates compile time error.]
- ○ This program generates runtime error.
- ○ This program generates output:
  java.lang.Number
  100
- ○ This program generates output:
  java.lang.Double
  10.5

Solution: This program generates compile time error because polymorphism is not applicable for generic type arguments.

3. Consider the Java code given below, and choose the correct option/s suitable for `Line 1`. [MSQ : 2 points]

```
public class Show{
    public <T> void display(T[] elements) {
         for (T element : elements){
            System.out.println(element);
         }
         System.out.println();
    }
}
public class Example{
   public static void main(String args[]) {
        ----------------------------- //Line 1
        Show obj1=new Show();
        obj1.display(arr1);
    }
}
```

- √ `Integer[] arr1 = {10, 20, 30, 40, 50};`
- √ `String[] arr1 = {"IIT","Madras","Java","Programming"};`
- ◯ `int[] arr1 = { 10, 20, 30, 40, 50 };`
- ◯ `double[] arr1= {20.5,30.7,56.6,67.8,0.25};`

Solution: The type variable `<T>` allows any type of array object.
Option 1: `Integer` object array can be passed to the type variable `<T>`.
Option 2: `String` object array can be passed to the type variable `<T>`.
Option 3: `int` primitive type array passed to the type variable `<T>` is invalid.
Option 4: `double` primitive type array passed to the type variable `<T>` is invalid.

4. Consider the code given below and choose the correct option.

[MCQ : 2 points]

```
public class Faculty {
    private  String name;
    private  String dept;
    public String getName() {
        return name;
    }
    public String getDept() {
        return dept;
    }
    public Faculty(String name, String dept) {
        this.name = name;
        this.dept = dept;
    }
    public String toString() {
        return "Faculty [name=" + name + ", dept=" + dept + "]";
    }
}
public class Hod extends Faculty {
    public Hod(String name, String dept) {
        super(name, dept);
    }
    public String toString() {
        return "Hod [name=" + getName() + ", dept=" + getDept() + "]";
    }
}
public class CopyArrayObjects {
    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) {
        Hod hod1 = new Hod("Johny", "CSE");
        Hod hod2 = new Hod("Jock", "EEE");
        Hod hod3 = new Hod("Nelson", "CE");
        Hod hod[] = {hod1,hod2,hod3};
        Faculty[] members = new Faculty[2];
        CopyArrayObjects.copy(hod, members);
        for (int i = 0; i < members.length; i++) {
```

```
        System.out.println(members[i]);
        }
    }
}
```

- [✓ This program generates output:
```
Hod [name=Johny, dept=CSE]
Hod [name=Jock, dept=EEE]
```
]
- ○ This program generates output:
```
Faculty [name=Johny, dept=CSE]
Faculty [name=Jock, dept=EEE]
```
- ○ This program generates output:
```
Hod [name=Johny, dept=CSE]
Hod [name=Jock, dept=EEE]
Hod [name=Nelson, dept=CE]
```
- ○ This program generates output:
```
Faculty [name=Johny, dept=CSE]
Faculty [name=Jock, dept=EEE]
Faculty [name=Nelson, dept=CE]
```
- ○ This code generates a compile time error.

Solution: While copying the arrays, the source array should be a subtype of the target array.

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

[MCQ:2 points]

```
public interface X{
   public abstract void display();
}
public class A{
   void show(){
      System.out.println("Show");
   }
}
public class B extends A implements X{
   public void display(){
      System.out.println("Display");
   }
}
public class Example<T extends A & X>{
   T obj;
   Example(T obj){
      this.obj=obj;
   }
   void show(){
      obj.display();
   }
}
public class Main{
   public static void main(String[] args){
      Example<B> c=new Example<B>(new B());
      c.show();
   }
}
```

- ○ This program generates output:
  Show
- [✓ This program generates output:
  Display]
- ○ This program generates output:
  Show
  Display
- ○ This program generates compile time error.

Solution: If A is a class and X is an interface than <T extends A & X> means that the type variable T can take any type arguments which is child class of A and implements interface x.

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

```
public class NumberFunction{
    public static <T extends Number> T max(T[] tArr){
        T max = tArr[0];
        for(int i = 0; i < tArr.length; i++) {
            if(tArr[i].doubleValue() > max.doubleValue()) {
                max = tArr[i];
            }
        }
        return max;
    }
}

public class FClass{
    public static void main(String[] args) {
        //LINE 1
        System.out.println(NumberFunction.max(arr));
    }
}
```

Identify the correct definition(s) for array `arr` for which the call `NumberFunction.max(arr)` can return the maximum element from the array `arr`.

- [✓ `Integer[] arr = {2, 4, 1, 6, 3};`]
- [✓ `Double[] arr = {2.3, 4.2, 1.4, 2.6, 1.3};`]
- ○ `Character[] arr = {'H', 'e', 'L', 'l', 'o'};`
- ○ `String[] arr = {"Apple", "test", "Apple", "Mango", "Orange"};`

Solution: For function `NumberFunction.max(arr)`, the type `T` is bounded by `Number`. Since `Integer` and `Double` both inherit from `Number`, function `NumberFunction.max(arr)` works correctly on them. However, it does not work for `Character` and `String`.

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

```
public interface Verifiable{
    public abstract boolean isEqual(Object d);
}
public class Employee implements Verifiable{
    private int id;
    private String name;
    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }
    public int get_id() {
        return id;
    }
    public String get_name() {
        return name;
    }
    public boolean isEqual(Object d) {
        if(d instanceof Employee)
            if(this.id == ((Employee)d).id)
                return true;
        return false;
    }
}
public class Manager extends Employee{
    private String department;
    public Manager(int id, String name, String department) {
        super(id, name);
        this.department = department;
    }
    public String get_department() {
        return department;
    }
}
public class FClass{
    //LINE 1: function-header
    {
        for(int i = 0; i < arr.length; i++) {
            if(m.isEqual(arr[i]))
                return true;
        }
        return false;
    }
    public static void main(String[] args) {
        Employee[] emps = {new Employee(101, "Darpan"),
            new Employee(102, "Aanya"), new Employee(103, "Binita"),
            new Employee(104, "Jairaj"), new Employee(105, "Ishaan")};
        Manager m = new Manager(103, "Binita", "IT");
        System.out.println(findEmployee(emps, m));
    }
}
```

Identify the appropriate function header for function `findEmployee(T[] arr, S m)`, such that the output is `true`

- [✓ `public static <T extends Verifiable, S extends T> boolean findEmployee(T[] arr, S m)`]
- ◯ `public static <T, S> boolean findEmployee(T[] arr, S m)`
- ◯ `public static <T, S extends T> boolean findEmployee(T[] arr, S m)`
- ◯ `public static <T extends Verifiable, T extends S> boolean findEmployee(T[] arr, S m)`

Solution: Since `Employee` implements `Verifiable` and `Manager` extends `Employee`, relation between `T` and `S` is:
`T extends Verifiable` and `S extends T`.

8. Consider the following code. [MSQ : 2points]

```
public class Example<T extends Number>{
  private T[] arr;
  public Example(T[] a){
    arr = a;
  }
}
public class ArrayObject {
  public static void main(String[] args) {
       -------Line 1--------
  }
}
```

Choose the correct option to fill in Line 1 to create an object of Array

- ◯ `Example<Integer> a = new Example<Integer>({1,2,3,4});`
- ◯ `int[] x = {1,2,3,4};`
  `Example<Integer> a = new Example<Integer>(x);`
- [✓ `Integer[] x = {1,2,3,4};`
  `Example<Integer> a = new Example<Integer>(x);`]
- ◯ `Example<Integer> a = new Example<Integer>( ){1,2,3,4};`
- ◯ `Example<String> a = new Example<String>({"one", "two", "three", "four"});`

Solution: `T` extends Number. So only Integer, Float, Double are compatible with `T`.
Option 1 is incorrect because of type mismatch.
Option 4 is incorrect because no constructor with zero arguments.

Consider the class `SampleClass` in the Java code given below, and answer the questions 9 and 10.

```
import java.lang.reflect.*;
public class SampleClass{
    private final int pr_data = 9;
    private String pr_str;
    public static int pu_data;
    private SampleClass() {
        //some code
    }
    public SampleClass(int pr_data_, String pr_str_) {
        pr_str = pr_str_;
    }
    public SampleClass(SampleClass tObj) {
        this.pr_str = tObj.pr_str;
    }
    private boolean isValid() {
        //some code
        return true;
    }
    public int get_pr_data() {
        return pr_data;
    }
    public String get_pr_str() {
        return pr_str;
    }
}
```

9. What should be the statements in `Line 1` and `Line 2`, respectively, such that the succeeding `for` loop prints the types of all the parameters of all the declared constructors in `SampleClass`?

```
public class FClass{
    public static void main(String[] args) {
        Class c = Class.forName("SampleClass");
        ________________________________________ //Line 1
        for(Constructor cont : my_const) {
            ________________________________________ //Line 2
            for(Class param : params) {
                System.out.print(param.getName() + ", ");
            }
        }
    }
}
```

Choose the correct option from below.

- ○ `Constructor[] my_const = c.getConstructors();`
  `Class params = cont.Name();`
- ○ `Constructors[] my_const = c.getAllConstructors();`
  `Class params[] = cont.getParameterTypes();`
- ○ `Constructor[] my_const = c.getDeclaredConstructors();`
  `Class param = cont.getMethodParameters();`
- √ `Constructor[] my_const = c.getDeclaredConstructors();`
  `Class params[] = cont.getParameterTypes();`
- ○ `Constructors[] my_const = c.getMehods();`
  `Class params[] = cont.Name();`

Solution: The solution follows from the syntax of the method in the class Class to obtain the declared constructors and the type of their parameters, of a given class.

10. If we have to print only the private instance variables of class SampleClass, which code snippet should we use?

◯
```
Field[] fields1 = c.getFields();
Field[] fields2 = c.getDeclaredFields();
for(Field f2 : fields2) {
    bool = Arrays.asList(fields1).contains(f2); //fields1 contains f2?
    if (bool == true) {
        pvt_fields[i] = f2.getName(); //add to private variables
        i = i + 1;
    }
}
for (j = 0; j < i; j++) {
    System.out.println(pvt_fields[j]);
}
```

◯
```
Field[] fields1 = c.getFields();
Field[] fields2 = c.getDeclaredFields();
for(Field f1 : fields1) {
    bool = Arrays.asList(fields2).contains(f1); //fields2 contains f1?
    if (bool == true) {
        pvt_fields[i] = f1.getName(); //add to private variables
        i = i + 1;
    }
}
for (j = 0; j < i; j++) {
    System.out.println(pvt_fields[j]);
}
```

◯
```
Field[] fields1 = c.getFields();
Field[] fields2 = c.getDeclaredFields();
for(Field f1 : fields1) {
    bool = Arrays.asList(fields2).contains(f1); //fields2 contains f1?
    if (bool == false) {
        pvt_fields[i] = f1.getName(); //add to private variables
        i = i + 1;
    }
}
for (j = 0; j < i; j++) {
    System.out.println(pvt_fields[j]);
}
```

✓
```
Field[] fields1 = c.getFields();
Field[] fields2 = c.getDeclaredFields();
for(Field f2 : fields2) {
    bool = Arrays.asList(fields1).contains(f2); //fields1 contains f2?
    if (bool == false) {
            pvt_fields[i] = f2.getName(); //add to private variables
            i = i + 1;
        }
    }
    for (j = 0; j < i; j++) {
        System.out.println(pvt_fields[j]);
    }
```

Solution: The code in option 4 checks for each declared field, whether it is present in the list of fields returned by `getFields()` method. If any is not present, then it is a private field, and it is added to the list of private fields. The first two options return public instance variables, and the third option does not return anything.