BSCCS2005: Graded Assignment with Solutions
Week 5

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

```
public class Dict<K, V> {
    private K key;
    private V value;

    public Dict(K key, V value) {
        this.key = key;
        this.value = value;
    }
}

public class Example {
  public static void main(String[] args) {
    ----------Line 1--------------
  }
}
```

Which of the following statements, if placed at `Line 1`, would create an object of Dict class?

- ○ `new Dict<Integer, Float>(1, 30);`
- ○ `new Dict<String, int>("John", 18);`
- √ `Dict<String, Integer> d = new Dict<String, Integer>("John", 18);`
- √ `new Dict<Integer, Integer>(1, 30);`

Solution: `K, V` are type parameters of generic types. `K, V` cannot be primitive types, so option b is incorrect.
In option 1 `V` is of type Float, but is initialized with int value which is not compatible here w.r.t generics.

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

[MCQ : 2 points]

```
public abstract class Vehicle{
    abstract void capacity();
}
public class Bike extends Vehicle{
    private String count="Bike capacity at most 2 persons";
    public void capacity() {
        System.out.println(count);
    }
}
public class Auto extends Vehicle{
    private String count="Auto capacity at most 4 persons";
    public void capacity() {
        System.out.println(count);
    }
}
public class Capacity {
    public <T extends Vehicle> void seating(T obj) {
        obj.capacity();
    }
    public static void main(String[] args) {
        Capacity bike=new Capacity();
        bike.seating(new Bike());
        Capacity auto=new Capacity();
        auto.seating(new Auto());
    }
}
```

- [✓ This program generates output:
  ```
  Bike capacity at most 2 persons
  Auto capacity at most 4 persons
  ```
  ]
- ◯ Compilation error at: `bike.seating(new Bike());`
- ◯ Compilation error at: `auto.seating(new Auto());`
- ◯ Compilation error at: `public <T extends Vehicle> void seating(T obj)`

Solution: Type parameter T extends from `Vehicle` class.
The type parameter can accept any subclass of `Vehicle`.

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

```
public class Player{
    private  String name;
    private  String type;
    public String getName() {
        return name;
    }
    public String getType() {
        return type;
    }
    public Player(String name, String type) {
        this.name = name;
        this.type = type;
    }
    public String toString() {
        return "Player [name=" + name + ", type=" + type + "]";
    }
}
public class Captain extends Player{
    public Captain(String name, String type) {
        super(name, type);
    }
    public String toString() {
        return "Captain [name=" + getName() + ", type=" + getType() + "]";
    }
}
public class CopyArrayObjects {
    public static ______________ void copy (S[] src, T[] tgt){    //LINE1
        int i,limit;
        limit = Math.min(src.length, tgt.length);
        for (i = 0; i < limit; i++){
            tgt[i] = src[i];
        }
    }
}
public class FClass{
    public static void main(String[] args) {
        Captain captain1=new Captain("Virat", "Batting");
        Captain captain2=new Captain("Hardik", "All Rounder");
        Captain captain3=new Captain("Jasprit", "Bowling");
        Captain captain[]= {captain1, captain2, captain3};
        Player[] player= new Captain[2];
        CopyArrayObjects.copy(captain,player);
        for (int i = 0; i < player.length; i++) {
            System.out.println(player[i]);
        }
    }
}
```

Identify the correct generic type parameter at `LINE 1` such that the given code prints the below text: Identify the correct generic type parameter at `LINE 1` such that the given code prints the below text:

```
Captain [name=Virat, type=Batting]
Captain [name=Hardik, type=All Rounder]
```

- ○ `<S, T>`
- ○ `<T super S, S>`
- ○ `<S, T extends S>`
- √ `<S extends T, T>`

Solution: Since the source array type (`S`) must extend target array type (`T`), option-4 is correct. Please note that `super` keyword is applicable only for the wildcard arguments.

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

```
public class NumData{
    private Number n;
    public NumData(Number n) {
        this.n = n;
    }
    public String getMetaInfo() {
        if (n instanceof Integer) {
            return "Integer type, value = " + n;
        }
        else if(n instanceof Double) {
            return "Double type, value = " + n;
        }
        else if(n instanceof Character) {
            return "Character type, value = " + n;
        }
        else
            return "Number type, value = " + n;
    }
}

public class FClass{
    public static void main(String[] args) {
        Integer i0 = 10;
        Float f0 = 3.14f;
        Character c0 = 'A';
        NumData o1 = new NumData(i0);
        NumData o2 = new NumData(f0);
        NumData o3 = new NumData(c0);
        System.out.println(o1.getMetaInfo());
        System.out.println(o2.getMetaInfo());
        System.out.println(o3.getMetaInfo());
    }
}
```

Choose the correct option regarding the given code.

- ◯ This program generates output:
  ```
  Integer type, value = 10
  Number type, value = 3.14
  Character type, value = A
  ```
- ◯ This program generates output:
  ```
  Integer type, value = 10
  ```

```
Double type, value = 3.14
Character type, value = A
```

- ◯ This program generates output:

```
Integer type, value = 10
Number type, value = 3.14
Number type, value = A
```

- √ This code generates compile time error because `Number` is not a super type of `Character`

Solution: All wrapper classes other than `Boolean`, `Character` extend the class `Number`.

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

```
public interface Iterator{
    public boolean has_next();
    public Object get_next();
}
public class NumList<T extends Number> implements Iterator{
    private T[] list;
    private int idx;
    public NumList(T[] list) {
        this.list = list;
        idx = 0;
    }
    public boolean has_next() {
        if(idx < list.length - 1)
            return true;
        return false;
    }
    public Object get_next() {
        idx++;
        return list[idx];
    }
}

public class FClass{
    ____________________________________________     //LINE 1: function-header
    {
        double total = 0;
        while(lOb.has_next()) {
            total += ((Number)lOb.get_next()).doubleValue();
        }
        return total;
    }
    public static void main(String[] args) {
        Integer[] i_arr = {10, 20, 30, 40, 59};
        Double[] d_arr = {3.44, 2.65, 6.44, 1.3, 6.78};
        NumList<Integer> i_list = new NumList<Integer>(i_arr);
        NumList<Double> d_list = new NumList<Double>(d_arr);
        System.out.println(sum(i_list) + ", " + sum(d_list));
    }
}
```

Identify the appropriate function header for function `sum`, such that the output is
`149.0, 17.17`

✓ `public static double sum(NumList<? extends Number> lOb)`

○ `public static double sum(NumList<Number> lOb)`

✓ `public static <T extends Number> double sum(NumList<T> lOb)`

○ `public static double sum(NumList<? super Number> lOb)`

Solution: In option-1, the parameter type `NumList<? extends Number>` is compatible with both `NumList<Integer>` and `NumList<Double>`. So, it is a correct option.
In option-2, the parameter type `NumList<Number>` is not compatible with `NumList<Integer>` and `NumList<Double>`. So, it is a wrong option.
In option-3, the parameter type `NumList<T>`, where the quntifier `T` is defined as `<T extends Number>`, is compatible with `NumList<Integer>` and `NumList<Double>`. Although, `T` is never used, it is a correct option.
In option-4, the parameter type `NumList<? super Number>` is not compatible with `NumList<Integer>` and `NumList<Double>` (`Integer` and `Double` are subtype of `Number`, not supertype). So, it is a wrong option.

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

```java
public class Employee{
    private String name;
    private double salary;
    public Employee(String name, double salary){
        this.name = name;
        this.salary = salary;
    }
    public String getName() {
        return name;
    }
    public double getSalary() {
        return salary;
    }
}
public class Developer extends Employee{
    //implementation with some new instance variable and methods
}
public class Manager extends Employee{
    //implementation with some new instance variable and methods
}
public class SalaryStat<T extends Employee>{
    private T[] eps;
    public SalaryStat(T[] eps) {
        this.eps = eps;
    }
    private double getTotalSalary() {
        double total = 0;
        for(int i = 0; i < eps.length; i++)
            total += eps[i].getSalary();
        return total;
    }
    public boolean greaterSalary(_______________) {    //LINE 1
        if (this.getTotalSalary() > d.getTotalSalary())
            return true;
        return false;
    }
}
public class FClass{
    public static void main(String[] args) {
        Developer[] dA = {new Developer("A", 50000.0), new Developer("B", 40000.0),
                new Developer("C", 45000.0)};
        Manager[] mA = {new Manager("X", 65000.0), new Manager("Y", 51000.0)};
        SalaryStat<Developer> dO = new SalaryStat<Developer>(dA);
        SalaryStat<Manager> mO = new SalaryStat<Manager>(mA);
        if(mO.greaterSalary(dO))
            System.out.println("managers have higher salary expenditure");
        else
            System.out.println("developers have higher salary expenditure");
    }
}
```

Identify the appropriate argument for function `greaterSalary`, such that the output is `developers have higher salary expenditure`

- ◯ `SalaryStat<T> d`
- √ `SalaryStat<?> d`
- ◯ `T d`
- √ `SalaryStat<? extends Employee> d`

Solution: Since the function `greaterSalary` compares the total salary of subclasses of `Employee`, the `LINE 1` either can be
`SalaryStatistic<? extends Employee>)`
or
`SalaryStatistic<?> d`, which matches any `SalaryStatistic` object.
For option 2 and 3, quantifier T is not defined.

Consider the class `SampleClass` in the Java code given below, and answer the questions 7 and 8.

```
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;
    }
}
```

7. What should be the statement in `Line 1` so that `Line 2` prints the number of all the public constructors in `SampleClass`?

[MCQ:2 points]

```
public class FClass{
    public static void main(String[] args) {
        Class c = Class.forName("SampleClass");
        ________________________________________ //Line 1
        System.out.println(my_const.length);     //Line 2
    }
}
```

Choose the correct option from below.

- ◯ `Constructor[] my_const = c.getMethods();`
- ◯ `Constructors my_const = c.getDeclaredConstructors();`
- ✓ `Constructor[] my_const = c.getConstructors();`
- ◯ `Constructor[] my_const = c.getDeclaredConstructors();`
- ◯ `Constructor my_const[] = c.getAllConstructors();`

Solution: The solution follows from the syntax of the method in the class Class to obtain the public constructors of a given class.

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

```
public class FClass{
    public static void main(String[] args) {
        Class c = Class.forName("SampleClass");
        Field[] fields1 = c.getFields();
        Field[] fields2 = c.getDeclaredFields();
        for(Field f : fields1) {
            System.out.print(f.getName() + " : ");
            System.out.print(f.getType());
            System.out.println();
            System.out.println("Modifier: " +
                    Modifier.toString(f.getModifiers()));
        }
        for(Field f : fields2) {
            System.out.print(f.getName() + " : ");
            System.out.print(f.getType());
            System.out.println();
            System.out.println("Modifier: " +
                    Modifier.toString(f.getModifiers()));
        }
    }
}
```

What will the output be?

○
```
pr_data :  int
Modifier:  private final
pr_str :  class java.lang.String
Modifier:  private
pu_data :  int
Modifier:  public static
```

✓
```
pu_data :  int
Modifier:  public static
pr_data :  int
Modifier:  private final
pr_str :  class java.lang.String
Modifier:  private
pu_data :  int
Modifier:  public static
```

○
```
pr_data :  int
Modifier:  private final
pr_str :  class java.lang.String
Modifier:  private
```

```
pu_data :  int
Modifier:  public static
pu_data :  int
Modifier:  public static
```

◯
```
pu_data :  int
Modifier:  public static
```

Solution: The first `for` loop prints all the public instance variables in the given class. The second `for` loop prints the public and private instance variables of the class. Only options 2 and 3 are printing both. But in option 3, the order is not correct. Hence the right answer is option 2.

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

```
public class Algorithm{
  public <T extends Integer> boolean findOdd(T a){
    if(a % 2 == 0){
      return false;
    }
    return true;
  }
  public <T> void display(T[] arr){
    for(T i: arr){
      System.out.println(i);
    }
  }
}
```

What is class `Algorithm` converted to after type erasure?

◯
```
public class Algorithm{
  public boolean findOdd(Number a){
    if(a % 2 == 0){
      return false;
    }
    return true;
  }
  public void display(Object[] arr){
    for(Object i: arr){
      System.out.println(i);
    }
  }
}
```

◯
```
public class Algorithm{
  public boolean findOdd(Object a){
    if(a % 2 == 0){
      return false;
    }
    return true;
  }
  public void display(Object[] arr){
    for(Object i: arr){
      System.out.println(i);
    }
  }
}
```

√
```
public class Algorithm{
  public boolean findOdd(Integer a){
    if(a % 2 == 0){
      return false;
    }
    return true;
  }
  public void display(Object[] arr){
    for(Object i: arr){
      System.out.println(i);
    }
  }
}
```

○
```
public class Algorithm{
  public boolean findOdd(T a){
    if(a % 2 == 0){
      return false;
    }
    return true;
  }
  public void display(T[] arr){
    for(T i: arr){
      System.out.println(i);
    }
  }
}
```

Solution: Type erasure replaces all type parameters in generic types with their bounds(super class) or Object if the type parameters are unbounded.

10. Consider the following code and choose the correct option regarding the same.

[MCQ:2points]

```
public interface Walkable{
    default void showPaceLength() {
        System.out.println("Average pace length : 0.4 meters");
    }
}
public class Human implements Walkable{
    double pace_length = 0.85;
    public void showPaceLength() {
        System.out.format("Average pace length : %f meters",pace_length);
    }
}

public class Mammal<T>{
    public String name;
    public T group;
    public Mammal(T obj){
        name = obj.getClass().getSimpleName();
        group = obj;
    }
    public void print() {
        System.out.println(name);
        group.showPaceLength();
    }
}
public class Test1 {
    public static void main(String[] args) {
        Mammal<Human> m = new Mammal<Human>(new Human());
        m.print();
    }
}
```

- ○ This code generates output: Human
  Average pace length : 0.850000 meters
- ○ This code generates output: Mammal
  Average pace length : 0.850000 meters
- ○ This code generates a compilation error because Human type can't be passed to the generic Mammal class.
- [✓ This code generates a compilation error because the method `showPaceLength` could not be resolved.]

◯ This code generates a compilation error because the method `getClass()` could not be resolved.