BSCCS2005: Practice Assignment with Solutions
Week 7

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

```
import java.util.logging.*;

public class FClass {
    public FClass() {
        Logger.getGlobal().config("cont called");
    }
    public void fun() {
        Logger.getGlobal().fine("fun() called");
    }
    public static void main(String[] args){
        FClass obj = new FClass();
        Logger.getGlobal().setLevel(Level.FINE);
        ConsoleHandler handler = new ConsoleHandler();
        handler.setLevel(Level.FINE);
        Logger.getGlobal().addHandler(handler);
        obj.fun();
        Logger.getGlobal().config("fun() return");
        Logger.getGlobal().log(Level.FINER, "end of main()");
    }
}
```

Identify the result when the code gets executed.

◯ It prints nothing

◯
```
<date time> FClass <init>
CONFIG: cont called
<date time> FClass fun
FINE: fun() called
<date time> FClass main
CONFIG: fun() return
<date time> FClass main
FINER: end of main()
```

√
```
<date time> FClass fun
FINE: fun() called
<date time> FClass main
CONFIG: fun() return
```

◯
```
<date time> FClass fun
FINE: fun() called
<date time> FClass main
CONFIG: fun() return
<date time> FClass main
FINER: end of main()
```

Solution: By default, three levels - `SEVERE`, `WARNING` and `INFO` - are logged and printed on the console. Thus, the string `"cont called"` is not logged.
After that, global logger is set log up to level `FINE`, and setting the level of `ConsoleHandler` object enables the logged data to be printed in console. Thus, the strings `FINE: "fun() called"` and `"CONFIG: fun() return"` would be logged as well as get printed on the console.
However, the string `"end of main()"` would not be logged, since level is set to `FINER` which is at lower than level `FINE`.

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

```
import java.util.logging.*;

public class SomeClass {
    private final static Logger logbook =
                        Logger.getLogger(SomeClass.class.getName());
    public void logIt(){
        logbook.info("logIt() called");
        logbook.setLevel(Level.INFO);
    }
}

public class FClass {
    private final static Logger logbook =
                        Logger.getLogger(FClass.class.getName());
    public static void main(String[] args){
        logbook.info("main() started");
        logbook.setLevel(Level.WARNING);
        SomeClass obj = new SomeClass();
        obj.logIt();
        logbook.info("main() ends");
    }
}
```

Identify the result when the code gets executed.

√
```
<date time> FClass main
INFO: main() started
<date time> SomeClass logIt
INFO: logIt() called
```

◯
```
<date time> FClass main
INFO: main() started
<date time> SomeClass logIt
INFO: logIt() called
<date time> FClass main
INFO: main() ends
```

◯
```
<date time> FClass main
INFO: main() started
```

◯
```
<date time> FClass main
INFO: main() started
<date time> PM FClass main
INFO: main() ends
```

Solution: The program logger two different logger for two different classes.
In class `FClass`, logger name is `FClass`. Since, by default, `INFO` level messages is logged, the message `"main() started"` is logged and printed. However, the next statement set the log level to `WARNING`, which is higher level than `INFO`. Thus, next message `main() ends"` would not be printed.
In class `SomeClass`, logger name is `SomeClass`. Since, by default, `INFO` level messages is logged, the message `"logIt() started"` is logged and printed.

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

```
public class MainClass{
    public static void main(String[] args){
        int a = 10;
        int b = 0;
        try{
            assert b != 0;
            int c = a / b;
        }
        catch(ArithmeticException e){
            System.out.println("in catch block");
        }
        finally{
            System.out.println("in finally block");
        }
        System.out.println("end of main()");
    }
}
```

Choose the most appropriate option regarding the code when executed as:
`java -ea MainClass`

- ◯ It only provides a runtime exception: `java.lang.AssertionError`
- √ It prints: `in finally block`,
  then provides a runtime exception: `java.lang.AssertionError`
- ◯ It prints:
  `in finally block`
  `end of main()`,
  then provides a runtime exception: `java.lang.AssertionError`
- ◯ It prints:
  `in catch block`
  `in finally block`
  `end of main()`

Solution: The `assert` statement in `try` block throws an `AssertionError`. However, there is no `catch` block to handle the exception. So it causes abrupt termination of the program. But, before termination it executes the `finally` block, then terminates after providing runtime exception information.

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

```
public class DOBRegistration{
    private int day, month, year;
    public DOBRegistration(int day, int month, int year){
        assert 0 < day && day <= 31: "day :" + day;      //assert-1
        this.day = day;
        assert 0 < month && month <= 12: "day :" + day;  //assert-2
        this.month = month;
        this.year = year;
    }
}
public class JobApplication{
    private int age, exp;
    public JobApplication(int age, int exp){
        assert age >= 18: "invalid age for job";         //assert-3
        this.age = age;
        assert exp >= 3: "invalid experience for job";   //assert-4
        this.exp = exp;
    }
}
public class FClass{
    public static void main(String[] args){
        DOBRegistration dr = new DOBRegistration(2, 23, 1879);
        JobApplication ja = new JobApplication(20, 1);
    }
}
```

Identify the `assert` statement that throws the `AssertionError` when the class is executed as:
`java -ea:JobApplication FClass`

- ◯ assert-1
- ◯ assert-2
- ◯ assert-3
- √ assert-4

Solution: Since assertions are enabled for class `JobApplication`, assert statement `assert-4` throws `AsserionError`.

5. Consider the following Java code and choose the correct option. [MCQ : 2 points]

```
public class Example{
    public static void main(String[] args) {
        int a[]=new int[10];
        try{
            a[10]=50;
            System.out.println("value in the array is "+a[10]);
        }
        catch (ArrayIndexOutOfBoundsException ae){
            System.out.println("Please check array index");
        }
        finally{
            System.out.println("end of main()");
        }
    }
}
```

◯ This program generates output:

```
value in the array is 50
end of main()
```

◯ This program generates output:

```
Please check array index
```

√ This program generates output:

```
Please check array index
end of main()
```

◯ This program generates output:

```
end of main()
```

Solution: In above program, the statement a[10] = 50; throws ArrayIndexOutOfBoundException. The corresponding catch block statements will be executed. After catch block, finally block statements will be executed.

6. Consider the following Java code and choose the correct option. [MSQ : 2 points]

```
public class Example{
    public static void main(String[] args) {
        java.util.Scanner scanner = new java.util.Scanner(System.in);
        int a,b,c;
        a=scanner.nextInt();
        b=scanner.nextInt();
        String name = "IIT madras java";
        int index=scanner.nextInt();
        try{
            c = a/b;
            System.out.println("Quotient is "+c);
            System.out.println(name.charAt(index));
        }
        catch(ArithmeticException | ArrayIndexOutOfBoundsException e){
            System.out.println(e);
        }
    }
}
```

If `b` is 0 and `index` is greater than the length of `name`, then what will the outcome be?

- ◯ The program terminates due to unhandled exception(s).
- √ It prints a message on `ArithmeticException`.
- ◯ It prints a message on whichever exception occurs first.
- ◯ It prints messages on both the exceptions.

Solution:

```
catch(ArithmeticException | ArrayIndexOutOfBoundsException e){
System.out.println(e);
}
```

Above is an alternative for the multi catch block. Both the exceptions can be handled. But only one exception can be handled at a time.

7. Consider the following Java code and choose the correct option.

[MCQ : 2 points]

```
public class A{
    public void show() throws Exception{
        System.out.println("Super class show with Exception");
    }
}
public class B extends A{
    public void show() throws RuntimeException{
        System.out.println("Sub class show with RuntimeException");
    }
}
public class Example {
    public static void main(String[] args) {
        B ob=new B();
        ob.show();
    }
}
```

◯ Compilation error because we cannot override the `show()` method in subclass B, as the `show()` method is throwing the more general exception, namely, `Exception` in super class A.

◯ Compilation error because we cannot override the `show()` method in subclass B, as the `show()` method is throwing the `RuntimeException`, which is not compatible with `Exception`.

√ This code generates the output:
`Sub class show with RuntimeException`

◯ This code generates the output:

```
Super class show with Exception
Sub class show with RuntimeException
```

Solution: You can override `show()` in subclass B by throwing RuntimeException. ob.show() will call `show()` from the subclass B.

8. Consider the following Java code and choose the correct option.

[MCQ : 2 points]

```
public class  Example{
    int show(){
        try{
            return 10;
        }
        catch (Exception e){
            return 20;
        }
        finally {
            return 100;
        }
    }
    public static void main(String[] args) {
        Example object=new Example();
        System.out.println(object.show());
    }
}
```

√ This code generates the output:
100

◯ This code generates the output:
20

◯ This code generates the output:
0

◯ Compilation failed.

Solution: Here, the try, catch and finally blocks are returning values to the method. In such a scenario, whatever value is returned by the finally block will be considered as the final return value.

9. Consider the following Java code and choose the correct option.

[MCQ : 2 points]

```
public class  Example{
    public void show() throws Exception{
        System.out.println("show() with Exception");
    }
    public void show(int a) throws RuntimeException{
        try {
            show();
        }
        catch (Exception e) {
        }
        System.out.println("show() overloaded with RuntimeException");
    }
    public void show(String a){
        show(10);
        System.out.println("show() overloaded with no Exceptions");
    }
    public static void main(String[] args){
        try{
            new Example().show("IITM Java");
        }
        catch (Exception e){
            System.out.println(e);
        }
    }
}
```

◯ Compilation error

◯ The program terminates abnormally because of ambiguity in the overloading of the show() method.

√ This program generates the output:

```
show() with Exception
show() overloaded with RuntimeException
show() overloaded with no Exceptions
```

◯ This program generates the output:

```
show() overloaded with no Exceptions
show() overloaded with RuntimeException
show() with Exception
```

Solution: Option 3 is correct
We can overload a method which throws `Exception`.
The overloading method may not throw any exception, or it may throw the same or a different exception as the original method.

10. Consider the following Java code and choose the correct way of using the class Example from any outside package.

[MSQ : 2 points]

```
package iitm.java.program;
public class  Example{
    public void show(){
        System.out.println("This is show");
    }
}
```

√
```
import iitm.java.program.Example;
public class UseExample {
    public static void main(String[] args){
        new Example().show();
    }
}
```

◯
```
public class UseExample {
    public static void main(String[] args){
        new Example().show();
    }
}
```

√
```
public class UseExample {
    public static void main(String[] args){
        new iitm.java.program.Example().show();
    }
}
```

◯
```
import iit.*;
public class UseExample {
    public static void main(String[] args){
        new Example().show();
    }
}
```

Solution: Options 1 and 3 are correct.
Option 1 imports the package using import keyword.
Option 3 imports the package using the fully qualified name.

11. Consider the following two source code files located in two different packages as shown.

[MSQ : 2 points]

//Adder.java

```
package iitm;
public class Adder {
    public int add(int n1, int n2, int n3) {
        return n1+n2+n3;
    }
    protected int add(int n1,int n2) {
        return n1+n2;
    }
}
```

//Test1.java

```
package test;
import iitm.*;

class Calculator extends Adder{
    public void calculate() {
        System.out.println(this.add(7,8,9)); // LINE 1
        System.out.println(this.add(9,10));  // LINE 2
    }
}
public class Test1{
    public static void main(String args[]) {
        Adder a1 = new Adder();

        System.out.println(a1.add(4,5));   // LINE 3
        System.out.println(a1.add(1,2,3)); // LINE 4

        new Calculator().calculate();
    }
}
```

Choose the correct option regarding these two .java files.

- ◯ LINE 1 will lead to compilation error.
- ◯ LINE 2 will lead to compilation error.
- √ LINE 3 will lead to compilation error.
- ◯ LINE 4 will lead to compilation error.

Solution: `protected` members of a class can only be accessed in subclasses within the package as well as outside the package. Members of a class where access specifier is no mentioned explictly is treated as *package private* types and can only be accessed within that specific package. `public` members are accessible through out all packages and all classes
Therefore the `Adder` object `a1` inside the `test1` class's `main` method can only access public members of `Adder` class.