Uttara InfoSolutions
www.uttarainfo.com

Exceptions Lab Document

Note: All tester classes, put it in a package named test, other classes declare in a package called com.uttara.ex.

1) Write a main() method program to generate and catch an ArrayIndexOutOfBoundsException. Test it without try catch. Code try..catch to catch this exception and print the stack trace. First have specific catch ArrayIndexOutOfBoundsException. Also have a SOP outside the try..catch.Is this executed? How come? Then replace the catch parameter to RuntimeException and see if the code still works. Change it to Exception and Throwable as well and see if it works. If the stack trace is printed when you do not have try..catch at all, why do we need try catch then?

2) WAP to generate the following abnormal conditions in main():

a) NullPointerException

b) ClassCastException

c) NumberFormatException

d) OutOfStackMemoryError (invoke main() recursively) -> can you catch this as well??

e) OutOfMemoryError - heap memory -> populate heap with 1 million reachable objects.

Enable a catch block for each and see whether you are able to catch the exceptions. Print the stack trace and verify.

3) Create an Animal class (empty body) and a subclass called Pig (empty body). Create a class called TextEx with a method m1(). In m1(), put an SOP(“m1()") and invoke m2() of same class. In m2(), invoke m3() as shown:

Code m3() like this:

```
public static void m3(Animal a)
{
        SOP(“in m3()”);
        Pig p  = (Pig) a;
}
public static void m2()
{
        SOP(“in m2()”);
        Animal a = new Animal();
```

```
            m3(a);
            Pig p = new Pig();
            m3(p);
        }
        public static void m1()
        {
            SOP(“in m1()”);
            m2();
        }
```

In main(), invoke m1(). Embed in try..catch, print ST. Identify bad code and fix it!

4)

Code a method m1() in class TestFinally. Let it return an int.

```
public int m1()
{
        int x = 10;
        try
        {
            System.out.println("in m1() x="+x );
            return ++x;
        }
        catch(Exception e)
        {
            System.out.println("in catch of m1() "+e.getMessage());
            return ++x;
        }
        finally
        {
            System.out.println("in finally() of m1() x = "+x);
            return ++x;
        }
}
```

Invoke this from main() by creating an object and print the returned value from m1() to monitor. Embed statements in main() in try..catch with printStackTrace() and a SOP. Verify what values are printed.

a) when try succeeds b) in try…write code to raise a NullPointerException and see what value gets printed c) in catch, create a NullPointerException and see what happens d) in finally, create a NullPointerException and see what happens e) remove the return from finally and re-execute scenarios a,b,c and d. What is the best practice you learn from this?

5) In main(), test the throw keyword usage.

```
try
{
        System.out.println(“going to use throws”);
```

```
        throw new NullPointerException("null pointer");
        System.out.println("after using throws");
}
catch(Exception e)
{
        System.out.println("in main()->catch() msg = "+e.getMessage());
        e.printStackTrace();
}
```

Will this compile?? change throws statement to this:

```
if(1==1)
        throw new NullPointerException("null pointer");
```

Compile, run the code and test it. Are you able to see that the NullPointerException is getting caught?

Now create a custom subclass of Exception like this:

```
//what do we call such classes?

public class MyException extends Exception
{
        //what should be coded in this class?
        public MyException()
        {
        }
        public MyException(String msg)
        {
                super(msg); // why do we do this?
        }
}
```

Now throw new MyException("this is my exception") instead of NPEx and then see how the program works. Is there any point in raising and catching our own exception?

6) Create a Person. A person has a name and an age (int). Code correct setter/getter methods with validations and test. Remember all setter and parameterised constructors must validate the invokers input and raise IllegalArgumentException if there is a validation failure.

```
Ex: public class Person {
        int age;
        public void setAge(int a)
        {
                if(a < 0 || a > 100)
        throw new IllegalArgumentException("Is this person born or is he already
dead?");
```

```
        }
        …
}
```

Create a Tester class, create a person object, set a bad age and see when you execute, are you getting the IllegalArgumentException or not? Do you understand why we should raise exceptions for validations?

7) To the person class, add a functionality called dance(). It should accept a string type as parameter. If age of person is > 50 and the type of the dance is “salsa”, then raise a business custom exception called DanceFailureException. When you throw DanceFailureException, compile the class. Does it compile? Now add throws clause to the method dance and see if it compiles. Now compile tester class and see if it compiles. Add a specific DanceFailureException catch and print the message. Do you understand the control flow?

8) Code an ATM class with withdraw() that accepts a Card reference and double amount as parameter. A Card has a account num (String) and balance (double). Implement all input validation checks and business validation checks in ATM class. Why do you code/ create custom expections?How to?What are checked exceptions? recap…ask doubts!

9) Create an Account. An Account has a balance (BigDecimal) and owner (String). An Account can be used to debit(BigDecimal amt), credit(BigDecimal amt) or checkBalance() that returns BigDecimal. Code the class with the following best practices:

a) hiding state b) exposing correct setters/getters with validation c) implement behaviours that impact state and are impacted by state d) business methods must throw checked exceptions for business failures