info@marwadiuniversity.ac.in

Marwadi UNIVERSITY

# Index

| | | | |
|---|---|---|---|
| Name. | Shashank Bagda | Year. | 2021 / 22 |
| Subject. | OOP | Class. | 2TK1 |
| Semester | 2 | Roll No. | 92100133020 |

| No. | Date | Experiment Description / Subject | Page | Marks | Faculty's Signature |
|---|---|---|---|---|---|
| 1 | 3/3/22 | Worksheet - 1 | | | [illegible] |
| 2 | 17/3 | Worksheet - 3 | | | [illegible] |
| 3 | 30/3 | Worksheet - 4 | | | [illegible] |
| 4 | 7/4 | Worksheet - 6 | | | [illegible] |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

Marwadi UNIVERSITY

Faculty of Technology (FOT) | Faculty of Architecture | Faculty of Commerce | Faculty of Computer Application | Faculty of Science | Faculty of Business Management | Faculty of Liberal Studies

Rajkot-Morbi Road, At & PO : Gauridad, Rajkot 360 003, Gujarat, India. T +91-281-2923112, 2924154 / 55 / 56 M : +91 97277 24661 / 62 / 63 / 64 / 65 / 66 | Email: info@marwadiuniversity.ac.in | www.marwadiuniversity.in

Subject Code: 01CT0105

Subject Name: Object Oriented Programming

B. Tech. Year – I (Semester II)

**UNIT – 6**

**Worksheet – 6**

Enrollment No: 92100133020

Name: Shashank Bagda

Subject Faculty:

Prof. Kapil Shukla

Information and Communication Technology Department,

Marwadi University

Q-1: Differentiate

Answer:

| Error | Exception |
|---|---|
| 1) Errorr mostly occur at run time that's they belong to an unchecked type. | 1) Exceptions are problems which can occur at runtime & compile time. |
| 2) It is mainly caused by application itself. | 2) It is mostly caused by environment in which application is running. |
| 3) Only checked exceptions are known to compiler. | 3) Exceptions won't be known to compiler. |
| 4) It belongs to java.lang.errorr package. | 4) It belongs to java.lang.exception. |
| 5) Example : Out of Memory Errorr. | 5) Example : Null pointer Exception. |

| Checked Exception | Unchecked Exception |
|---|---|
| 1) It occurs at compile time | 1) Occurs at Runtime. |
| 2) These type of exceptions can be handled at the time of compilation. | 2) These type of exceptions can't be catch or handle at the time of compilation, because they get mistakes generated in the program. |
| 3) The compiler can check a checked exception. | 3) The compiler does not check these exception. |
| Ex : Filenot found Exception | Ex : Arithmetic Exception. |

Q-2: Explain following keywords with example:

a) try:

The keyword troy is also known as troy block. Troy block is used to enclose the code that might throow an exception. It must be used within the method. If an exception occuros at the paroticulan statement in the troy block, the roest of the block code will not execute. So it recommended not to keep the code in troy block that will not throow an exception. A code can have multiple troy. We cannot wroite troy without catch and finally.

Eg:
```
class Test
{
  public static void main (Stroing []arogs)
  {
    troy
    {
        int data = 10/0 ;
    }
    catch (Aroithmetic Exception)
    {
        System. out. proint ln (e);
    }
} }
```

b) catch:

In some case more than one exception could be raised by single piece of code. To handle this type of situation, we can specify two or more catch. ~~Each~~ Each catching a different type of exception. When an exception is thrown, each catch statement is inspected in order.

Code :
```
class Test
{
  public static void main (String []args)
  {
    try
    {
      int data = 10/0 ;
      System.out.println ("Addition");
    }
    catch (ArithmeticException)
    {
      System.out.println (e);
    }
  }
}
```

c) finally:

Java finally block used to ~~execute~~ important ~~more~~ code such as closing the connection, etc. Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception.

Code :
```
class Test
{
  public static void main (String [] args)
  {
    try
    { int data = 25/5 ;
      System.out.println (data);
    }
    catch (Null Pointer Exception e)
    {
      System.out.println (e);
    }
    finally {
    system.out.println ("finally block"); } } }
```

**d) throw:**

In some case more than one exception could be raised by single piece of code. To handle this type of situation, we can specify two or more catch. Each catching a different type of exception. When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed.

Code :
```
class Test {
public static void validate (int age )
{  if (age <18)
   {
    throw new ArithmeticException ("Person is not eligible");
   }
   else
   { System.out.println ("Person is eligible");
   }
  }
```

**e) throws:**

If method is capable of causing an exception that it does not handle, it must specify this behaviour so that caller of the method can guard themselves against the exception.

Code :
```
class Test
{
   static void throwOne ()
   {
   System.out.println ("Inside throwOne");
   throw new IllegalAccessException ("demo");
   }
   public static void main (String args[])
   {
   throwOne ();
   }
}
```

Q-3: Explain Nested try ... catch block with example.

Answer:

The try statement can be nested. It can be inside the block of another try. Each time a try statement is entered the context of that exception is pushed on the stack. In an inner try statement does not have a catch handler for a particular exception. The stack is unwound and the next try statement catch handler are inspected for a match. If no catch statement matches then Java run-time system will handle the exception.

Code:

```
class Test
{
  public static void main (String []args)
  {
    try
    {
    try
    {
    try
    {  int arr [] = (1, 2, 3, 4);
       system. out. println (arr [10]);
    }
    catch (Arithmetic Exception e)
    {
       System.out.println ("Arithmetic Exception by try block 3");
    }
    catch (Arithmetic Exception e)
    { System.out.println ("Arithmetic Exception by try block 2");
    }
    catch (Arithmetic Exception e)
    {
       System.out.printth("Arithmetic Exception by try block 1");
    }
  }
}
```

Q-4: Explain Custom exception with example.

Answer:

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Java custom exception are used to customize the exception according to user need. Using the custom exception, we can have our own exception and message. We use custom to catch and provide specific treatment to a subset of existing Java exception. In order to create custom exception, we need to extend exception class that belongs to java.lang package.

Code :
```
class MyException extends Exception
{
  private int detail;
  MyException (int a)
  {
    detail = a;
  }
  public String to String()
  {
    return "My Exception [" + detail + "]";
  }
}
class ExceptionDemo
{
  Static void compute (int a) throws MException
  {
    System.out.println ("called compute (" + a + ")");
    if (a>10)
    throw new Exception (a);
    System.out.println ("Normal exist");
  }
  public static void main (String args[])
  {
    try
    {
```
[illegible]

| References | | |
|---|---|---|
| Q No | Book Name | Page No |
| 1 | Java 2 The complete reference & Java 8 black book | 175-184 & 109 |
| 2 | Java 8 black book | 132 |
| 3 | Java 2 | 179 |
| 4 | Java 2 / Java 8 Black Book | 187 / 139 |

```
        compute (1);
        compute (20);
      }
      catch (Exception e)
      {
        System.out.println ( " caught" + e);
      }
    }
  }
```