Fundamentals of Object Oriented Programming

CSN- 103

Dr. R. Balasubramanian

Associate Professor

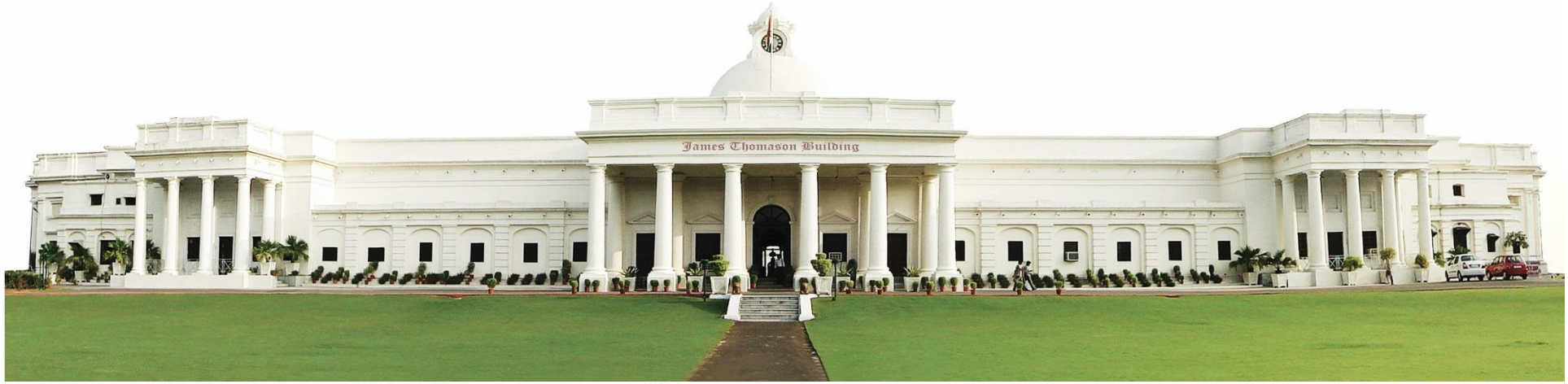
Department of Computer Science and Engineering

Indian Institute of Technology Roorkee

Roorkee 247 667

balarfcs@iitr.ac.in

https://sites.google.com/site/balaiitr/



protected in Java

```java
//CSN-103, IITR
class Bike{
 int speedlimit=80;
}
class Honda3 extends Bike{
// int speedlimit=160;

 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj);
  System.out.println(obj.speedlimit);
}
}
```

Terminal

```
sh-4.3$ javac Honda3.java
sh-4.3$ java Honda3
Honda3@659e0bfd
80
sh-4.3$
```

- **http://goo.gl/qSJyto**

```java
//CSN-103, IITR
class Bike{
 private int speedlimit=80;
}
class Honda3 extends Bike{
// int speedlimit=160;

 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj);
  System.out.println(obj.speedlimit);
}
}
```

Terminal

```
sh-4.3$ javac Honda3.java
Honda3.java:11: error: speedlimit has private access in Bike
  System.out.println(obj.speedlimit);
                         ^
1 error
sh-4.3$
```

- **http://goo.gl/GkwsLc**

protected

```java
//CSN-103, IITR
class Bike{
 protected int speedlimit=80;
}
class Honda3 extends Bike{
 //int speedlimit=160;

 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj);
  System.out.println(obj.speedlimit);
}
}
```

Terminal

```
sh-4.3$ javac Honda3.java
sh-4.3$ java Honda3
Honda3@659e0bfd
80
sh-4.3$
```

- **http://goo.gl/kfCQJX**

Access Modifiers (Revisit)

- Lecture 35: Access Modifiers, Method overriding, Run Time Polymorphism (25/10/18)

Role of Private Constructor

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

default access modifier

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
//save by B.java
 package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  } }
```

protected access modifier

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

protected access modifier

```java
//save as A.java
//CSN-103, IITR
package pack;
public class A{
protected void msg(){System.out.println("Hello I am Protected");}
}
```

http://goo.gl/FTwFJy

```java
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

Terminal

```
sh-4.3$ javac -d . A.java
sh-4.3$ javac -d . B.java
sh-4.3$ java mypack.B
Hello I am Protected
sh-4.3$
```

codingground
SIMPLYEASYCODING

Compile and Execute Java8 Online

New Project

- root
  - A.java
  - B.java
  - HelloWorld.java
  - mypack
    - B.class
  - pack
    - A.class

Compile | Execute | Share Code | HelloWorld.java × | A.java ×

```java
//save as A.java
//CSN-103, IITR
package pack;
public class A{
protected void msg(){System.out.println("Hello I am Protected");}
}
```

Default Term | + | Browser

```
sh-4.4$ javac -d . A.java
sh-4.4$ javac -d . B.java
sh-4.4$ java mypack.B
Hello I am Protected
sh-4.4$
```

public access modifier

- Done in Lecture 35, Slide nos. 4 and 5

Visibility Modifiers

Accessible to:	public	protected	Package (default)	private
Same Class	Yes	Yes	Yes	Yes
Class in package	Yes	Yes	Yes	No
Subclass in different package	Yes	Yes	No	No
Non-subclass different package	Yes	No	No	No

Java access modifiers with method overriding

- If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```java
class A{
protected void msg(){System.out.println("Hello I am Protected in IITR");}
}

public class Simple extends A{
void msg(){System.out.println("Hello I am not Protected");}
 public static void main(String args[]){
   Simple obj=new Simple();
   obj.msg();
   }
}
```

Terminal

```
sh-4.3$ javac Simple.java
Simple.java:6: error: msg() in Simple cannot override msg() in A
void msg(){System.out.println("Hello I am not Protected");}
     ^
  attempting to assign weaker access privileges; was protected
1 error
sh-4.3$
```

http://goo.gl/cLsHtt

Interface in Java

- An **interface in java** is a blueprint (a design plan) of a class. It has static constants and abstract methods only.

- The interface in java is **a mechanism to achieve fully abstraction**.

- There can be only abstract methods in the java interface not method body.

- It is used to achieve fully abstraction and multiple inheritance in Java.

Why use Java interface?

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.

interface

```
interface Printable{
int MIN=5;
void print();
}
```

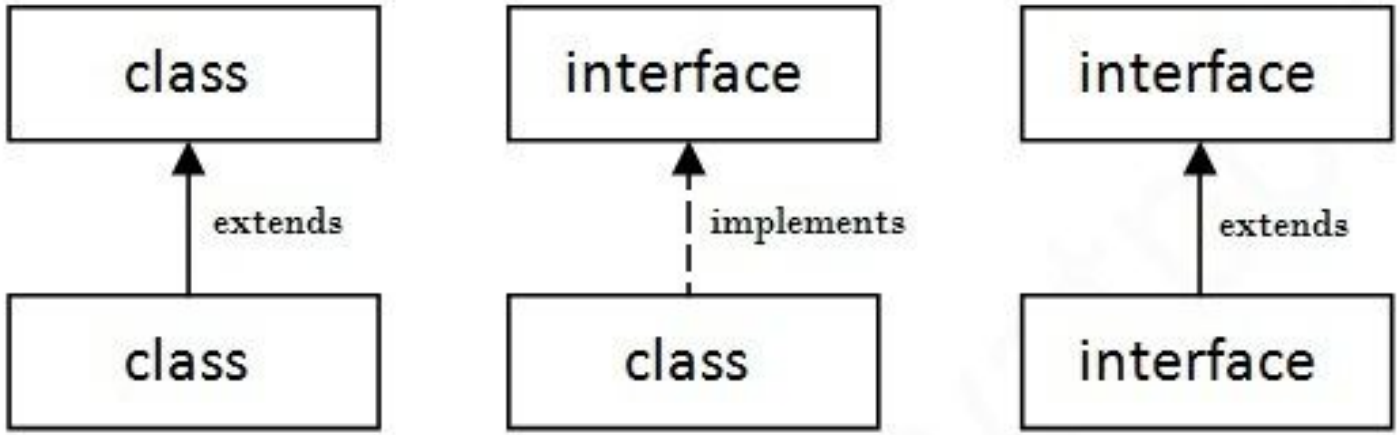
Printable.java

compiler

```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

Printable.class

Understanding relationship between classes and interfaces



interface

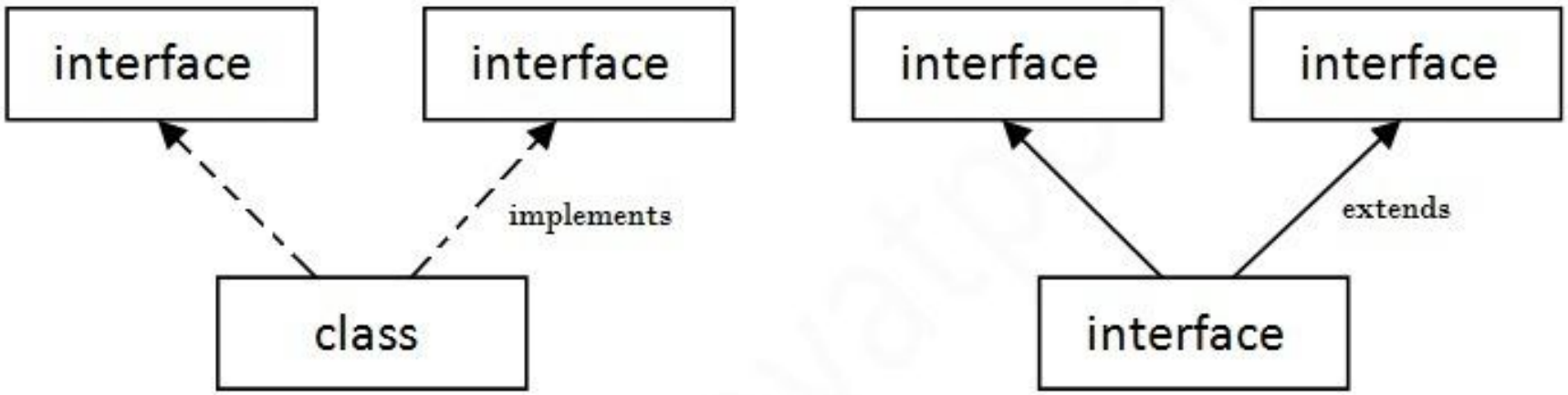
```java
interface printable{
void print();
}

class A6 implements printable{
public void print(){System.out.println("Learning Interface @ IITR");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Terminal

```
sh-4.3$ javac A6.java
sh-4.3$ java A6
Learning Interface @ IITR
sh-4.3$
```

http://goo.gl/ugxxLg

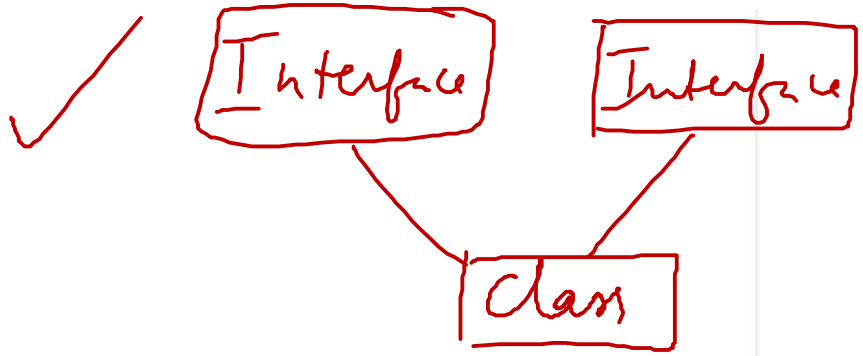
Multiple inheritance in Java by interface



Multiple Inheritance in Java

Multiple inheritance in Java by interface

```java
interface Printable{
void print();
}

interface Showable{
void show();
}

class A7 implements Printable,Showable{

public void print(){System.out.print("Hi, We are learning");}
public void show(){System.out.println(" Multiple Inheritance through JAVA interface @ IITR");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```



Terminal

```
sh-4.3$ javac A7.java
sh-4.3$ java A7
Hi, We are learning Multiple Inheritance through JAVA interface @ IITR
sh-4.3$
```

- **http://goo.gl/NooHDS**

- Multiple inheritance is not supported through class in java but it is possible by interface, why?
- it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

Multiple inheritance

```java
interface Printable{
void print();
}

interface Showable{
void print();
}

class A8 implements Printable,Showable{

public void print(){System.out.println("CSN-103 @ IITR");}

public static void main(String args[]){
A8 obj = new A8();
obj.print();
}
}
```

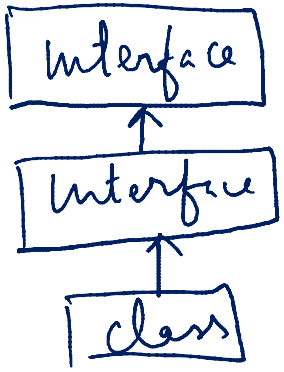
Terminal

```
sh-4.3$ javac A8.java
sh-4.3$ java A8
CSN-103 @ IITR
sh-4.3$
```

- **http://goo.gl/vmCk75**

Interface inheritance

```java
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class Testinterface2 implements Showable{

public void print(){System.out.print("Hi, We are learning");}
public void show(){System.out.println(" Interface inheritance @ IITR");}

public static void main(String args[]){
Testinterface2 obj = new Testinterface2();
obj.print();
obj.show();
 }
}
```



Terminal

```
sh-4.3$ javac Testinterface2.java
sh-4.3$ java Testinterface2
Hi, We are learning Interface inheritance @ IITR
sh-4.3$
```

http://goo.gl/x4pQPF

Nested Interface in Java

- An interface can have another interface i.e. known as nested interface.

```
interface printable{
 void print();
 interface MessagePrintable{
   void msg();
 }
}
```

Difference between abstract class and interface

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.

Difference between abstract class and interface

5) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

7) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Exception Handling in Java

- The **exception handling in java** is one of the powerful *mechanisms to handle the runtime errors* so that normal flow of the application can be maintained.

- In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Introduction:

- Errors in coding are categorised as:
  - **syntax errors –**

    compilation errors.
  - **Semantic/logical errors–**

    leads to programs producing unexpected outputs.
  - **runtime errors –**

    most often lead to abnormal termination of programs or even cause the system to crash.

Common Runtime Errors:

- Dividing a number by zero.
- Accessing an element that is out of bounds of an array.
- Trying to store incompatible data elements.
- Using negative value as array size.
- Trying to convert from string data to a specific data value (e.g., converting string “abc” to integer value).
- File errors:
- opening a file in “read mode” that does not exist or no read permission
- Opening a file in “write/update mode” which has “read only” permission.
- .....

Without Error Handling – Example

```
class NoErrorHandling{
        public static void main(String[] args){
                int a,b;
                a = 7;
                b = 0;
                System.out.println("Result is " + a/b);
                System.out.println("Program reached this line");
        }
}
```

Program does not reach here

No compilation errors. While running it reports an error and stops Without executing further statements:
java.lang.ArithmeticException: / by zero at Error2.main(Error2.java:10)

Advantage of exception handling

- The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
```

Exceptions

- What are they?
  - An exception is a representation of an error condition or a situation that is not the expected result of a method.
  - Exceptions are built into the Java language and are available to all program code.
  - Exceptions isolate the code that deals with the error condition from regular program logic.

Common Java Exceptions

- ArithmeticException
- ArrayIndexOutOfBoundException
- ArrayStoreException
- FileNotFoundException
- IOException – general I/O failure
- NullPointerException – referencing a null object
- OutOfMemoryException
- SecurityException – when applet tries to perform an action not allowed by the browser's security setting.
- StackOverflowException
- StringIndexOutOfBoundException

Types of Exception

- There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception.

1) Checked Exception

- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions
- e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

- The classes that extend RuntimeException are known as unchecked exceptions
- e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

Checked Exception

Execute | Embed main.cpp Stdin

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World" << endl;
    int a[10];
    cout<<a;
    return 0;
}
```

```
$g++ -o main *.cpp
$main
Hello World
0x7ffcc338a480
```

Checked Exceptions

Execute | Embed main.cpp Stdin

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World" << endl;
    int a[1000000000000000];
    cout<<a;
    return 0;
}

```

```
$g++ -o main *.cpp
$main
timeout: the monitored command dumped core
sh: line 1: 88295 Bus error               timeout 10s main
```

Common scenarios where exceptions may occur

- **1) Scenario where ArithmeticException occurs**

```
int a=50/0;//ArithmeticException
```

```java
public class ArithExcep{

    public static void main(String []args){
        int a=50/0;
        System.out.println(a);
    }
}
```

Terminal

```
sh-4.3$ javac ArithExcep.java
sh-4.3$ java ArithExcep
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ArithExcep.main(ArithExcep.java:4)
sh-4.3$
```

http://goo.gl/l07yXG

2) Scenario where NullPointerException occurs

```java
public class NULLPtrExc{

    public static void main(String []args){
        String s=null;
        System.out.println(s.length());//NullPointerException
    }
}
```

Terminal

```
sh-4.3$ javac NULLPtrExc.java
sh-4.3$ java NULLPtrExc
Exception in thread "main" java.lang.NullPointerException
        at NULLPtrExc.main(NULLPtrExc.java:5)
sh-4.3$
```

- **http://goo.gl/OORdJB**

3) Scenario where NumberFormatException occurs

```java
public class FormatExcept{

    public static void main(String []args){
        String s="abc";
        int i=Integer.parseInt(s);//NumberFormatException
        System.out.println(i);
    }
}

```

Terminal

```
sh-4.3$ javac FormatExcept.java
sh-4.3$ java FormatExcept
Exception in thread "main" java.lang.NumberFormatException: For input string: "abc"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.parseInt(Integer.java:615)
        at FormatExcept.main(FormatExcept.java:5)
sh-4.3$
```

http://goo.gl/xWOJK6

4) Scenario where ArrayIndexOutOfBoundsException occurs

```java
public class ArrayExcept{

    public static void main(String []args){
        int a[]=new int[5];
        a[10]=50; //ArrayIndexOutOfBoundsException
        System.out.println(a[10]);
    }
}
```

Terminal

```
sh-4.3$ javac ArrayExcept.java
sh-4.3$ java ArrayExcept
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at ArrayExcept.main(ArrayExcept.java:5)
sh-4.3$
```

http://goo.gl/z8OnuC

Java Exception Handling Keywords

- There are 5 keywords used in java exception handling.
  - `try`
  - `catch`
  - `finally`
  - `throw`
  - `throws`

Exception-Handling:

- Java exception handling is managed by via five keywords: **try, catch, throw, throws**, and **finally.**
- Program statements to monitor are contained within a **try block.**
- If an exception occurs within the **try block**, it is thrown.
- Code within **catch block** catch the exception and handle it.
- System generated exceptions are automatically thrown by the Java run-time system.
- To manually throw an exception, use the keyword **throw.**
- Any exception that is thrown out of a method must be specified as such by a **throws** clause.

Uncaught Exceptions

- If an exception is not caught by user program, then execution of the program stops and it is caught by the default handler provided by the Java run-time system
- Default handler prints a stack trace from the point at which the exception occurred, and terminates the program

Ex:

```
class Exc0 {
  public static void main(String args[]) {
    int d = 0;
    int a = 42 / d;
  }
}
```

Output:

```
java.lang.ArithmeticException: / by zero
      at Exc0.main(Exc0.java:4)
Exception in thread "main"
```

Uncaught Exceptions

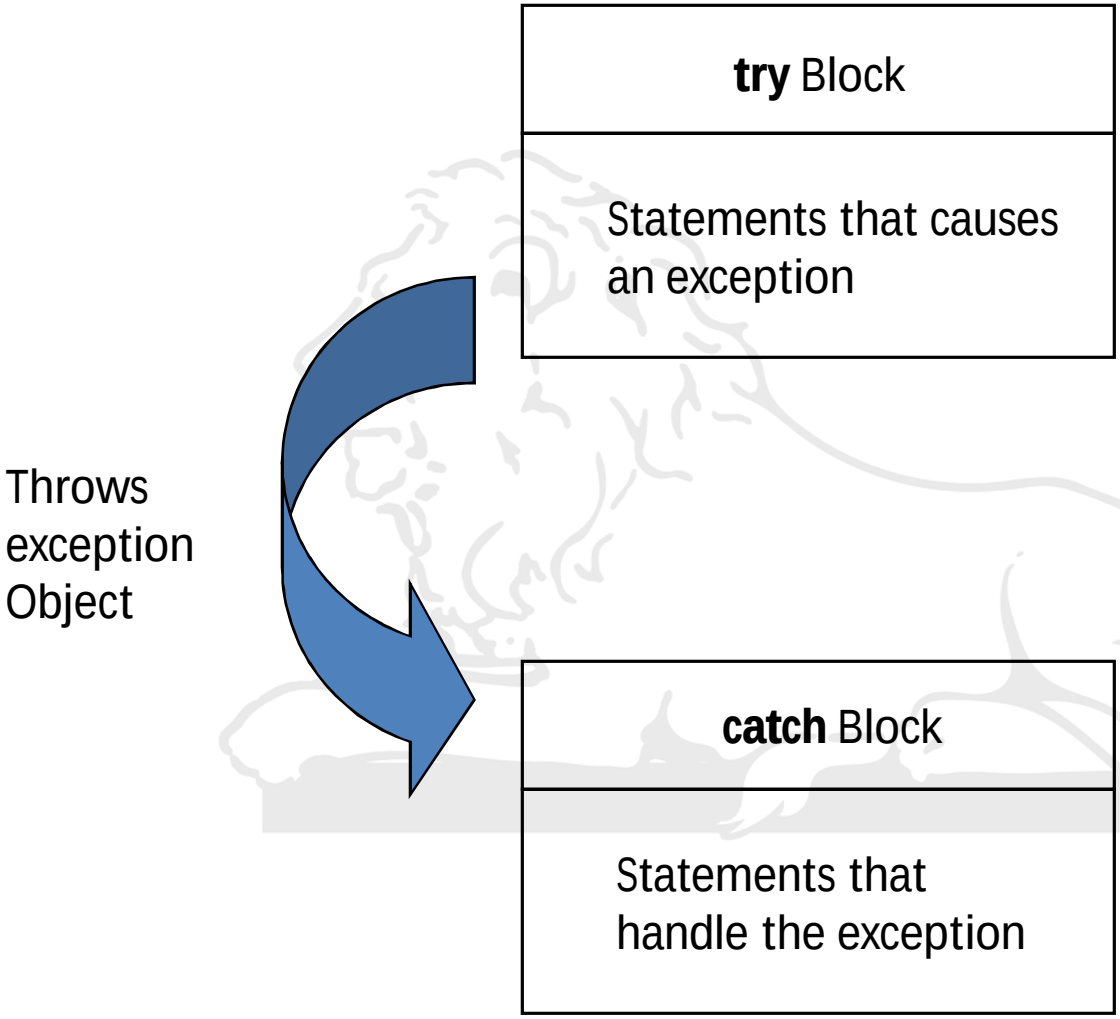
Ex:

```
class Exc1 {
static void subroutine() {
int d = 0;
int a = 10 / d;
}
public static void main(String args[]) {
Exc1.subroutine();
}
}
```

- **Output: IS THIS CORRECT?**

```
java.lang.ArithmeticException: / by zero
    at Exc0.main(Exc0.java:4)
Exception in thread "main"
```

Exception Handling Mechanism



Syntax of Exception Handling Code

```
…
…
try {
      // statements
}
catch( Exception-Type e)
{
     // statements to process exception
}
..
..
```

Use of try catch

```java
public class Testtrycatch1{
  public static void main(String args[]){
      int data=50/0;//may throw exception
      System.out.println("rest of the code...");
}
}
```

Terminal

```
sh-4.3$ javac Testtrycatch1.java
sh-4.3$ java Testtrycatch1
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Testtrycatch1.main(Testtrycatch1.java:3)
sh-4.3$
```

http://goo.gl/RVMzju

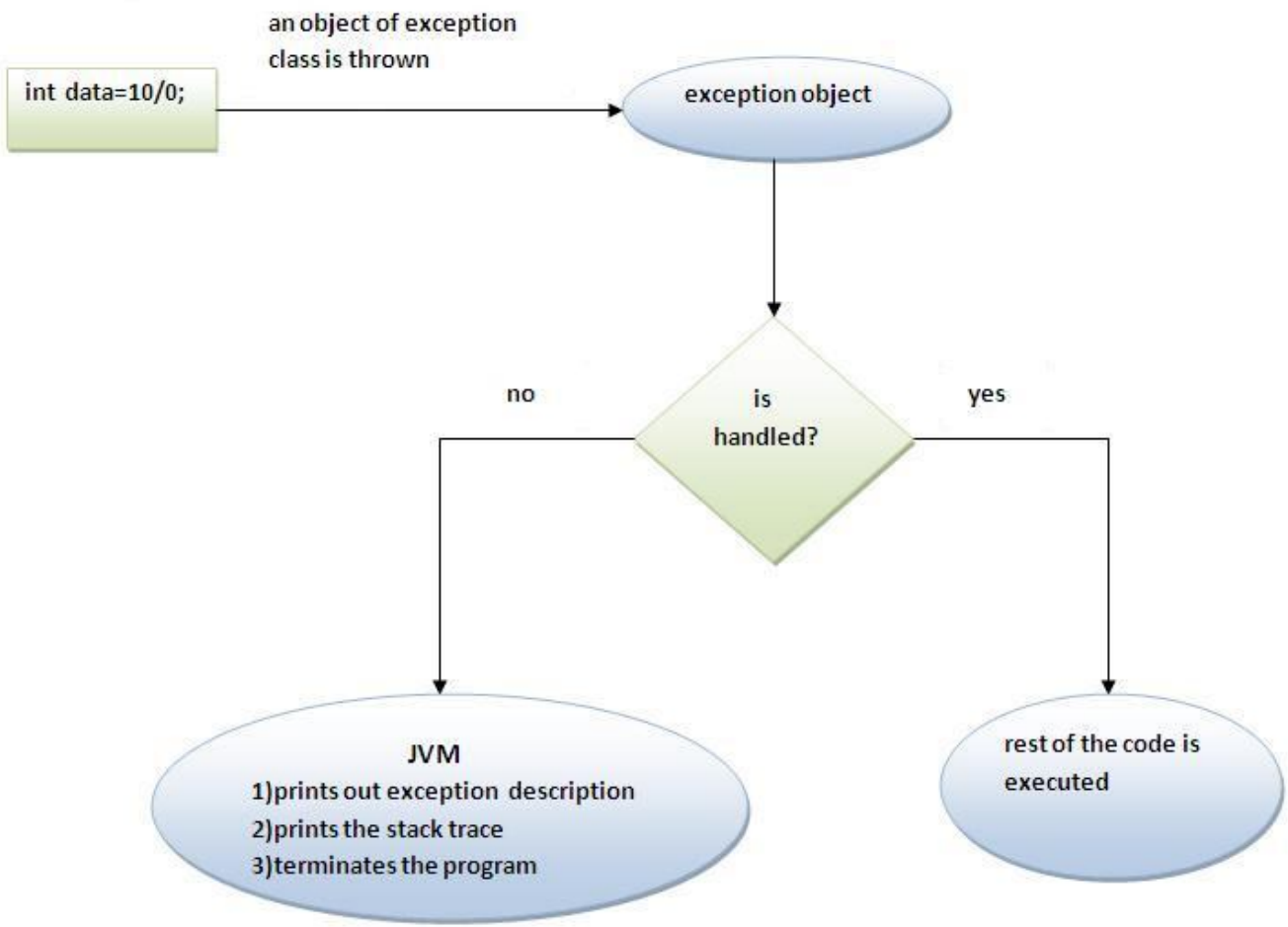
```java
public class Testtrycatch2{
  public static void main(String args[]){
   try{
      int data=50/0;
   }
   catch(ArithmeticException e)   // catch( ) {....}
   {System.out.println(e);}
   System.out.println("rest of the code...");
}
}
```

Terminal

```
sh-4.3$ javac Testtrycatch2.java
sh-4.3$ java Testtrycatch2
java.lang.ArithmeticException: / by zero
rest of the code...
sh-4.3$
```

- **http://goo.gl/RVMzju**

Internal working of java try-catch block



Multiple catch Clauses

- In some cases, more than one exception could be raised by a single piece of code.
- To handle this type of situation, you can specify two or more catch clauses, 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.
- After one catch statement executes, the others are bypassed, and execution continues after the try/catch block.
- The example given in next slide traps three different exception types:

Multiple Catching in JAVA

```java
public class TestMultipleCatchBlock{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=100/0;
   }
   catch(ArithmeticException e){System.out.println("task1 is completed");}
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
   catch(Exception e){System.out.println("common task completed");}

   System.out.println("rest of the code...");
  }
}
```

Terminal

```
sh-4.3$ javac TestMultipleCatchBlock.java
sh-4.3$ java TestMultipleCatchBlock
task1 is completed
rest of the code...
sh-4.3$
```

- **http://goo.gl/jnZ92x**

Multiple Catching in JAVA

- At a time only one Exception is occurred and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception.

Multiple Catching in JAVA

```java
class TestMultipleCatchBlock1{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=30/0;
   }
   catch(Exception e){System.out.println("common task completed");}
   catch(ArithmeticException e){System.out.println("task1 is completed");}
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
   System.out.println("rest of the code...");
  }
}
```

Terminal

```
sh-4.3$ javac TestMultipleCatchBlock1.java
TestMultipleCatchBlock1.java:8: error: exception ArithmeticException has already been caught
   catch(ArithmeticException e){System.out.println("task1 is completed");}
   ^
TestMultipleCatchBlock1.java:9: error: exception ArrayIndexOutOfBoundsException has already been caught
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
   ^
2 errors
sh-4.3$
```

- **http://goo.gl/LVxb0m**


Thank you all
so much!!!
www.ungiornonellavita.com