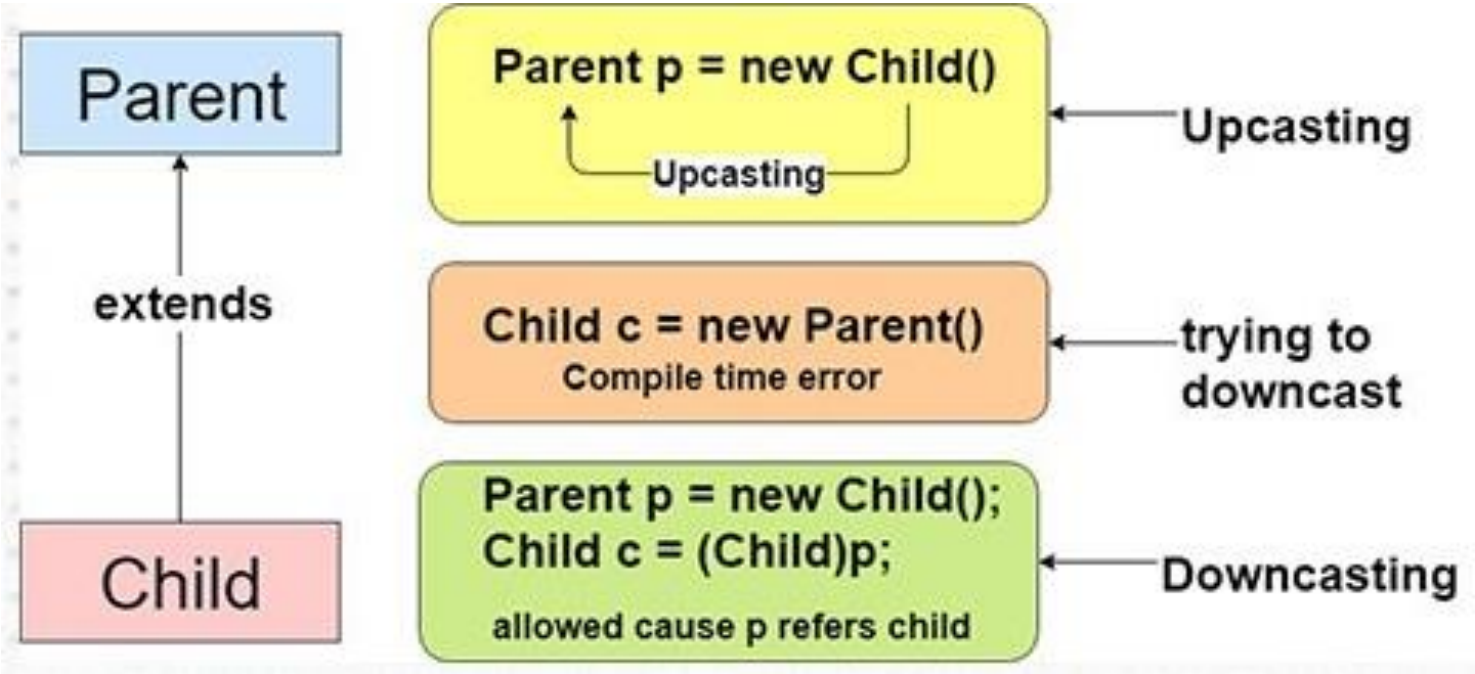
Object Oriented Programming
CS F213
Amit Dua

BITS Pilani
Pilani Campus

Slides Taken from the slides prepared by Dr. Jennifer

Upcasting and down casting



Upcasting

```
class Parent{
    void show() { System.out.println("Parent's show()"); }}
class Child extends Parent{
        void show() { System.out.println("Child's show()"); } }
class Main {
    public static void main(String[] args)
    {
        Parent obj1 = new Child();
        obj1.show();
    }   }
```

Downcasting

```
class Parent{
 void show(){
System.out.println("in Parent"); }}
class Child extends Parent{
void show(){
System.out.println("in overridden
    child"); }
void showSpecific(){
System.out.println("in specific child");
    }}
```

```
public class Main{
public static void
    main(String[] args) {

Parent p1 = new Child();

p1.show();
if(p1 instanceof Child){

Child c1 = (Child)p1;

c1.showSpecific();   } } }
```

Access Modifiers

innovate achieve lead

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

private

```
package p1;
class A {
   private void display(){
        System.out.println("Simple Snippets");    }}
class B {
   public static void main(String args[]){
        A obj = new A();
        //trying to access private method of another class
        obj.display();      }
}
```

Default access modifier

```
package p1;
class MyClass1 {//Class MyClass1 is having Default access modifier
    void display()       {
          System.out.println("Hello World!");} }
package p2;
import p1.*;
class MyClass2 {
    public static void main(String args[])       {
        //accessing class MyClass1 from package p1
        MyClass1 obj = new MyClass1();
        obj.display();       } }
```

Protected

```
package p1;
public class A {
   protected void display()     {
        System.out.println("Simple Snippets");     } }
package p2;
import p1.*; //importing all classes in package p1
class B extends A {
   public static void main(String args[])    {
       B obj = new B();
       obj.display();      }     }
```

innovate achieve lead

BITS Pilani

Pilani Campus

Exception handling

Uncaught Exceptions

- When the Java run time detects an exception, it constructs an object and throws the exception.
- If there are no exception handlers in the program, the exception is caught by the default exception handler.
- The default handler display a string describing the exception and prints a stack trace from the point at which the exception occurred and terminates the program.

Example

```
package class1;

public class test  {
public static void main(String args[]) {
int b = 30,c = 0;
System.out.println(b/c);
}
}
```

Console:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at class1.test.main(test.java:5)
```

Using try and catch

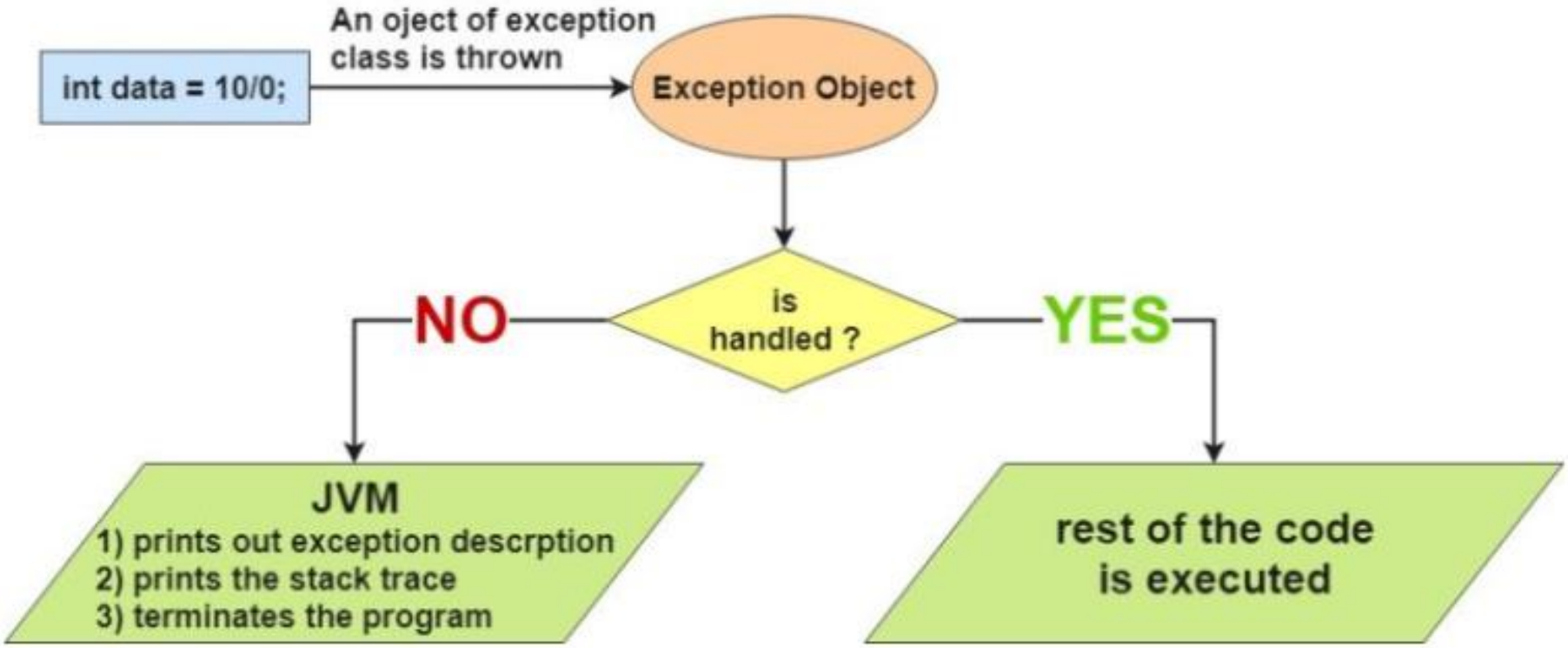
- Enclose the code you want to monitor inside the try block and include a catch block immediately after the try block
- Once an exception is thrown, the program control transfers out of the try block into the catch and it never returns to the try block
- A try and its catch is a unit. The scope of the catch is restricted to the statements specified by the immediately preceding try statement.
- A catch statement cannot catch an exception thrown by another try statement.

Try and catch

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

```
Exception in thread main java.lang.ArithmeticException:/ by
    zero
rest of the code...
```

Internal working of java try-catch block

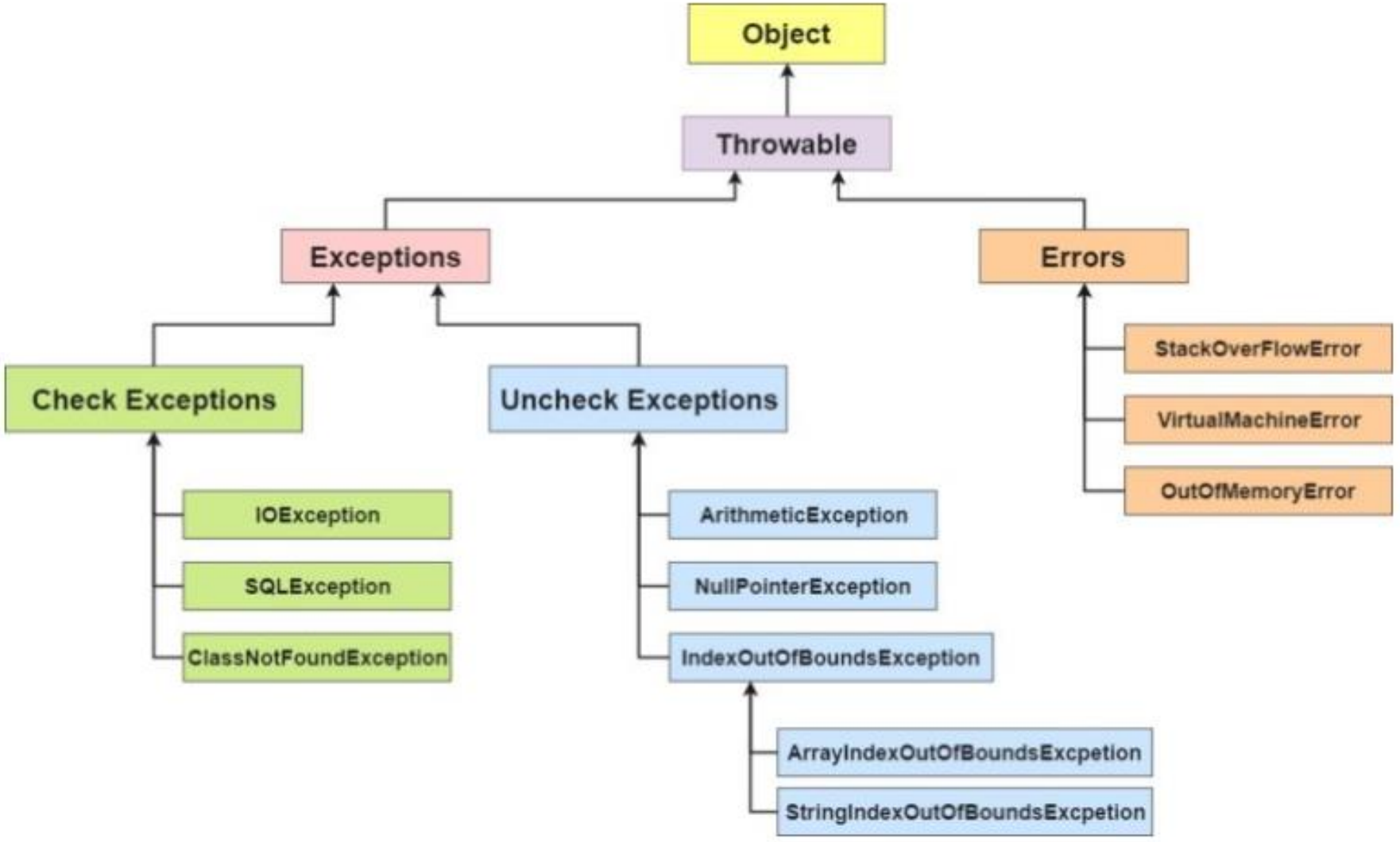


try catch finally

```
class TryCatchFinallyExample {
    public static void main (String[] args){
 int[] arr = new int[4];
try        {
 int i = arr[4];
// this statement will never execute
 // as exception is raised by above
    statement
 System.out.println("Inside try block"); }
catch(ArrayIndexOutOfBoundsException
    ex){
System.out.println("Exception caught in
    catch block"); }
finally
      {
            System.out.printl
n("finally block
executed");
        }

        // rest program will
be executed
        System.out.println("
Outside try-catch-finally
clause");
    }
}
```

Hierarchy of Java Exception classes



Key Concepts

innovate achieve lead

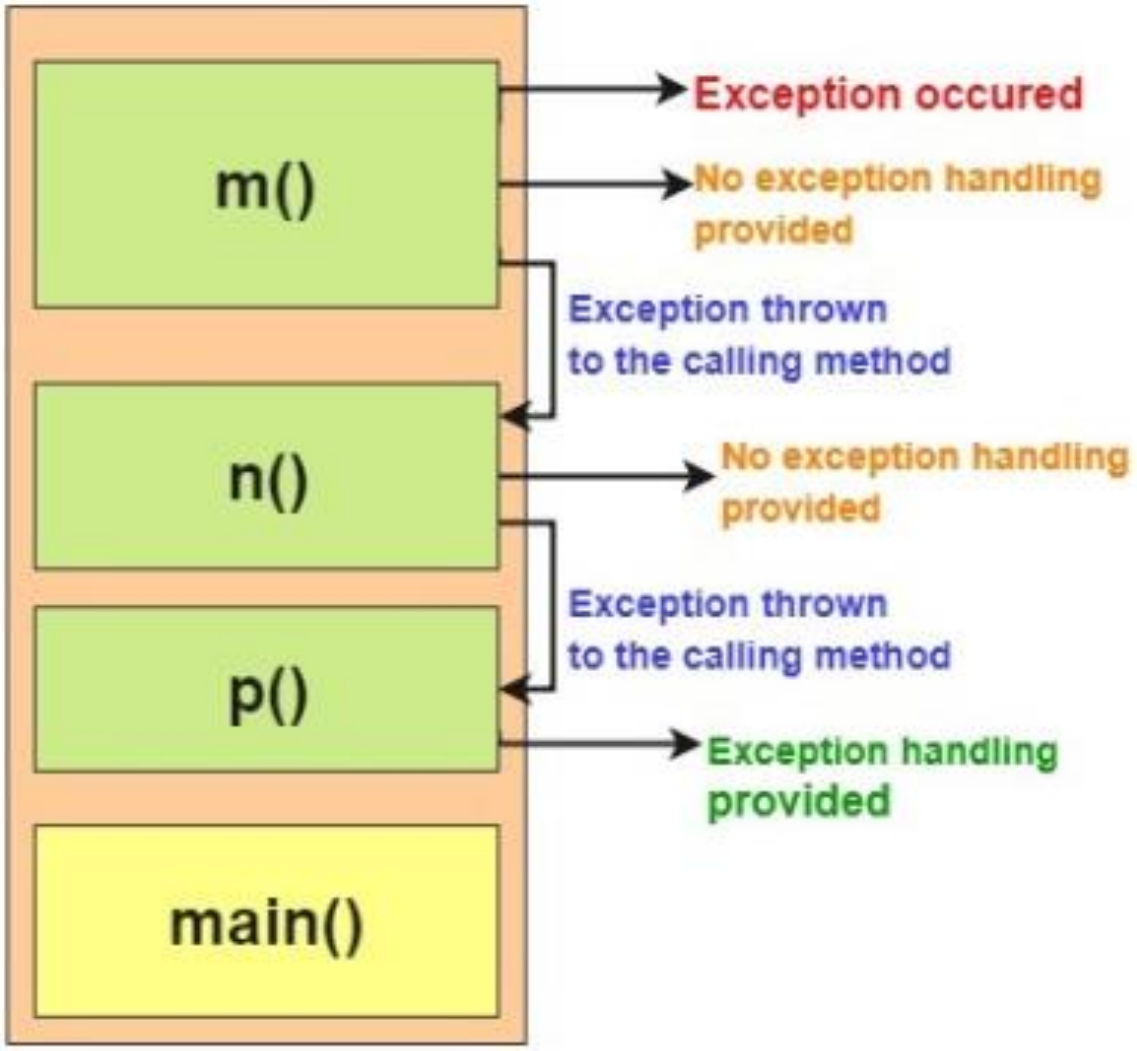
- Displaying an exception
  - Throwable overrides the toString(), and the exception object can be passed to println() and the description of the exception gets printed.
- Multiple catch clauses
  - A single try block can be accompanied with multiple catch blocks, but the exception subclass should come before any of their super classes.
- Nested try
  - If the inner try statement does not have a catch handler for a particular exception the next try statement's catch handler are inspected for a match.
  - Nested try can occur less obvious ways when method calls are involved.

Multiple try-catch

```
public class TestMultipleCatchBlock{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=30/0;     }
   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...");
}
```

```
class TestExceptionPropagation1{
  void m(){       int data=50/0;    }
  void n(){       m();     }
  void p(){
   try{       n();
   }catch(Exception e){System.out.println("exception
    handled");}    }
  public static void main(String args[]){
   TestExceptionPropagation1 obj=new
    TestExceptionPropagation1();
   obj.p();
   System.out.println("normal flow...");     }   }
```

Memory Stack



Key Concepts

- ‘throw’ keyword is used to throw an exception explicitly.
- But object thrown should be of type Throwable or subclass of throwable. Primitive types such as int, char … and non throwable classes like String can not be used.
- Many built in run-time exceptions have atleast two constructors, one that takes a string parameter describing the exception and one no argument constructor.
  - e.getMessage() can also be used.

Throw

```
public class TestThrow1{
   static void validate(int age){
     if(age<18)
      throw new ArithmeticException("not valid");
     else
      System.out.println("welcome to vote");
   }
   public static void main(String args[]){
      validate(13);
      System.out.println("rest of the code...");
  }
}
```

Key Concepts

- ‘throws’ keyword is used to specify the callers of the method, when it is capable of causing an exception and it does not handle.

- ‘finally’ is executed after try/catch block and before the code following the try/catch block whether or not an exception is thrown.
  - finally clause is optional
  - But every try block should have atleast one catch or finally block.
  - Even if there is a return statement in the try block, the method returns after the finally block is executed.

throws

```
import java.io.IOException;
class Testthrows1{    void m()throws IOException{
 throw new IOException("device error");//checked
    exception     }
  void n()throws IOException{       m();     }
  void p(){      try{     n();
   }catch(Exception e){System.out.println("exception
    handled");}     }
  public static void main(String args[]){
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");     }   }
```

Throw vs throws

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Throw is followed by an instance.	Throws is followed by class.
3)	Throw is used within the method.	Throws is used with the method signature.
4)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Extra Topic (only if you are interested) Difference between C++ and JAVA

- In C++, all types (including primitive and pointer) can be thrown as exception. But in Java only throwable objects (Throwable objects are instances of any subclass of the Throwable class) can be thrown as exception.
- In C++, there is a special catch called “catch all” that can catch all kind of exceptions. In Java, we can catch Exception object to catch all kind of exceptions.
- In Java, there is a block called finally that is always executed after the try-catch block. This block can be used to do cleanup work.
- In C++, all exceptions are unchecked. In Java, there are two types of exceptions – checked and unchecked.
- In Java, a new keyword *throws* is used to list exceptions that can be thrown by a function. In C++, there is no *throws* keyword, the same keyword *throw* is used for this purpose also.