Exceptions

Exception

- An exception is an object that defines an unusual or erroneous situation.
- can be caught and handled if desired

- An error is similar to an exception except that an error generally represents an unrecoverable situation and should not be caught.

Examples of situations that cause exceptions to be thrown

- attempting to “divide by zero”
- accessing an out-of-bound array element
- a specified file not found
- trying to invoke a method on null reference

How to deal with Exceptions ?

3 ways:

- **do not handle the exception at all**
- **handle the exceptions where it occurs**
- **handle the exception at another point in the program**

Advantage of Exception Model

- Most important advantage is:

  Separating Error Handling Code from "Regular" Code

  (we will see that in a moment…)

Uncaught Exceptions

- **if a program does not handle the exception at all, then:**
  - **it will terminate abnormally**
  - **produce a message that describes what exception occurred and where it originated**

Uncaught Exceptions

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

        int a = 100;  int b = 0;

        int c = divide( a, b );
        System.out.println( "Answer is: " + c);
    }

    public static int divide( int i, int j ) {
         return i / j;
    }
}
```

Output ??

Output of DivideByZero program

- **java DivideByZero**

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
   at DivideByZeroTest.divide(DivideByZero.java:11)
   at DivideByZeroTest.main(DivideByZero.java:6)
```

Catching and Handling Exceptions (where it occurs)

```
public class DivideByZeroTest {
   ……

  public static int divide( int i, int j ) {
     int k = 0;
     try { k = i / j; }
     catch( ArithmeticException ex ) {
       System.out.println( “Caught exception.” );
     }
     return k;
  }
}
```

```
public class DivideByZeroTest {
  public static void main(String[] args) {
     int a = 100; int b = 0; int c = divide(a, b);
     System.out.println( "Answer is: " + c);
  }
  public static int divide( int i, int j ) {
     int k = 0;
     try { k = i / j; }
     catch( ArithmeticException ex ) {
       System.out.println( "Caught exception." );
     }
     return k;
  }
}
```

Output:

Caught exception.

Answer is: 0

```
public class DivideByZeroTest {
  public static void main(String[] args) {
     int a = 100; int b = 10; int c = divide(a, b);
     System.out.println( "Answer is: " + c);
  }
  public static int divide( int i, int j ) {
     int k = 0;
     try { k = i / j; }
     catch( ArithmeticException ex ) {
       System.out.println( "Caught exception." );
     }
     return k;
  }
}
```

Output:

Answer is: 10

Example

```
public class ProductCodes {

  public static void main(String[] args) {

    String code = “2000R”;
    int district;
    boolean banned = false;
    char zone;
```

```
try {
    zone = code.charAt( 4 );
    String s = code.substring( 0, 4 );
    district = Integer.parseInt( s );
    if( zone == 'R' && district > 2000 )
        banned = true;
}
catch( StringIndexOutOfBoundsException e){
    System.out.println("Improper code length");
}
catch( NumberFormatException e){
    System.out.println("District not numeric");
}
```

```
    System.out.println("Code banned: " + banned);
  }
}
```

```
String code = "2000R";
```

Output:

Code banned: false

```
String code = "a345";
```

Output:

Improper code length

Code banned: false

```
String code = "a345P";
```

Output:

District not numeric

Code banned: false

```
String code = "a345P";
```

Output:

Code banned: true

Finally clause

- **A try-catch statement can have an optional finally clause.**
- **A finally clause is executed no matter how the try block is exited.**
- **used to manage resources (e.g. file close)**

Finally clause

- **If no exception is generated, the statements in finally clause are executed after the try block is complete**

- **If exception is generated in try block, control first tranfers to appropriate catch clause. After executing the catch clause, control goes to finally clause.**

- **A finally clause (if present) must be listed after catch clauses.**

- **A try block must have catch clause(s) OR finally clause OR both.**

Exception – try finally (exception not handled)

```
public class TestTryFinally {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        finally {
            System.out.println("inside finally");
        }
        System.out.println("program ends");
    }
}
```

Output??

Exception – try finally (exception not handled)

```
public class TestTryFinally {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        finally {
            System.out.println("inside finally");
        }
        System.out.println("program ends");
    }
}
```

Output:

inside finally

Exception – try catch finally (exception handled)

```
public class TestTryCatchFinally1 {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        catch( ArrayIndexOutOfBoundsException e ) {
            System.out.println("inside catch");
        }
        finally {
            System.out.println("inside finally");
        }
        System.out.println("program ends");
    }
}
```

Output??

Exception – try catch finally (exception handled)

```
public class TestTryCatchFinally1 {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        catch( ArrayIndexOutOfBoundsException e ) {
            System.out.println("inside catch");
        }
        finally {
            System.out.println("inside finally");
        }
        System.out.println("program ends");
    }
}
```

Output:

```
inside catch
inside finally
program ends
```

Exception – try catch finally (exception not handled)

```
public class TestTryCatchFinally2 {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        catch( NullPointerException e ) {
            System.out.println("inside catch");
        }
        finally {
            System.out.println("inside finally");
        }
        System.out.println("program ends");
    }
}
```

Output??

Exception – try catch finally (exception not handled)

```
public class TestTryCatchFinally2 {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        catch( NullPointerException  e ) {
            System.out.println("inside catch");
        }
        finally {
            System.out.println("inside finally");
        }
        System.out.println("program ends");
    }
}
```

Output:

inside finally

Exception – try catch (exception handled)

```
public class TestTryCatch1 {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        catch( ArrayIndexOutOfBoundsException  e ) {
            System.out.println("inside catch");
        }

        System.out.println("program ends");
    }
}
```

Output??

Exception – try catch (exception handled)

```
public class TestTryCatch1 {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        catch( ArrayIndexOutOfBoundsException  e ) {
            System.out.println("inside catch");
        }

        System.out.println("program ends");
    }
}
```

Output

```
inside catch
program ends
```

Exception – try catch (exception not handled)

```
public class TestTryCatch2 {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        catch( NullPointerException  e ) {
            System.out.println("inside catch");
        }

        System.out.println("program ends");
    }
}
```

Output??

Exception – try catch (exception not handled)

```
public class TestTryCatch2 {
    public static void main(String[] args) {
        int[] array = { 3 };
        try {
            int j = array[1];
            System.out.println( j );
        }
        catch( NullPointerException  e ) {
            System.out.println("inside catch");
        }

        System.out.println("program ends");
    }
}
```

Output

No output in this case

Handling exceptions at another point in a program

- If an exception is not caught and handled where it occurs, control is immediately returned to the method that invoked the method that produced the exception (called exception propagation).

- Propagation continues until the exception is handled or until it is passed out of main method, which terminates the program and prints the information about the exception.

Handling exceptions at another point in a program

- **To catch an exception at an outer level, the method that produces the exception must be invoked inside a try block that has catch clauses to handle it.**

Exception Propagation: Example1

```
public class TestPropagation {
    public static void main(String[] args) {
        try {
            TestPropagation tp = new TestPropagation();
            tp.test(2, 0);
        }
        catch( ArithmeticException  e ) {
            System.out.println("inside catch in main method");
        }
    }
    public int test( int i, int j ) {
        int k = i / j;
        System.out.println( k );
        return k;
    }
}
```

Output??

Exception Propagation: Example1

```
public class TestPropagation {
    public static void main(String[] args) {
        try {
1           TestPropagation tp = new TestPropagation();
2           tp.test(2, 0);
            System.out.println("end of try in main");
        }
        catch( ArithmeticException  e ) {
4           System.out.println("inside catch in main");
        }
    }
    public int test( int i, int j ) {
3       int k = i / j;
        System.out.println( k );
        return k;
    }
}
```

Output

inside catch in main

Exception Propagation: Example2

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

        ExceptionScope demo = new ExceptionScope();
        System.out.println("Program beginning");

        demo.level1();

        System.out.println("Program ending");

    }
}
```

```
public class ExceptionScope {
    public void level1() {
        System.out.println("level1 beginning");
        try {
            level2();
        }
        catch( ArithmeticException e ) {
            System.out.println( "message: " + e.getMessage() );
            System.out.println( "call stack trace: " );
            e.printStackTrace();
        }
        System.out.println("level1 ending");
    }
```

```
    public void level2() {
        System.out.println("level2 beginning");
        level3();
        System.out.println("level2 ending");
    }

    public void level3() {
        System.out.println("level3 beginning");
        int i = 10; int j = 0;
        int result = i / j;
        System.out.println("level3 ending");
    }
}
```

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

1   ExceptionScope demo = new ExceptionScope();
2   System.out.println("Program beginning");

3   demo.level1();

        System.out.println("Program ending");

    }
}
```

```
public class ExceptionScope {
    public void level1() {
4       System.out.println("level1 beginning");
        try {
5           level2();
        }
        catch( ArithmeticException e ) {
            System.out.println( "message: " + e.getMessage() );
            System.out.println( "call stack trace: " );
            e.printStackTrace();
        }
        System.out.println("level1 ending");
    }
```

```
    public void level2() {
6       System.out.println("level2 beginning");
7       level3();
        System.out.println("level2 ending");
    }

    public void level3() {
8       System.out.println("level3 beginning");
9       int i = 10; int j = 0;
10      int result = i / j;
        System.out.println("level3 ending");
    }
}
```

exception occurs

level2 does not handle this exception, so control goes to level1

```
public class ExceptionScope {
    public void level1() {
4       System.out.println("level1 beginning");
        try {
5           level2();
        }
        catch( ArithmeticException e ) {
11          System.out.println( "message: " + e.getMessage() );
12          System.out.println( "call stack trace: " );
13          e.printStackTrace();
        }
14      System.out.println("level1 ending");
    }
```

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

1   ExceptionScope demo = new ExceptionScope();
2   System.out.println("Program beginning");

3   demo.level1();

15  System.out.println("Program ending");

    }
}
```

Output of Example2

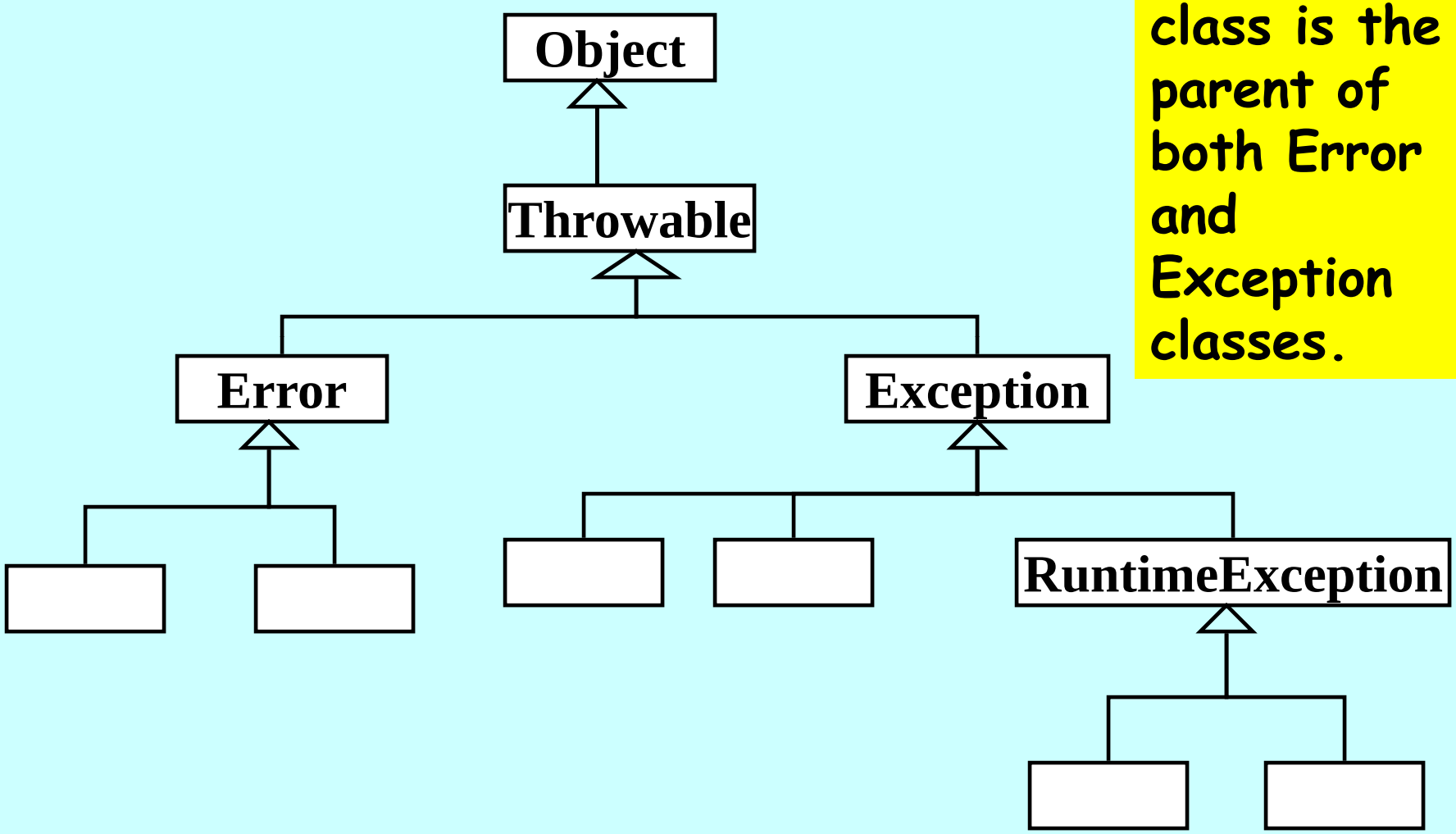
```
Program beginning
level1 beginning
level2 beginning
level3 beginning
message: / by zero
call stack trace:
java.lang.ArithmeticException: / by zero
        at ExceptionScope.level3(ExceptionScope.java:24)
        at ExceptionScope.level2(ExceptionScope.java:17)
        at ExceptionScope.level1(ExceptionScope.java:5)
        at Propagation.main(Propagation.java:7)
level1 ending
Program ending
```

Exception Class Hierarchy (Classes related by inheritance)

- **Throwable class is the parent of both Error and Exception class.**

- **we can define our own exceptions by deriving a new class from Exception class or one of its descendants. (gives it the ability to be thrown by the throw statement)**

Part of the Exception Class Hierarchy (Classes related by inheritance)



Throwable class is the parent of both Error and Exception classes.

Defining an exception

```
public class OutOfRangeException extends Exception {

    public OutOfRangeException( String message ) {
        super( message );
    }

    public OutOfRangeException() {
        super();
    }
}
```

we can define our own exceptions by deriving a new class from Exception class or one of its descendants. (gives it the ability to be thrown by the throw statement)

throw statement

- exceptions can be thrown using **throw** statement.

- e.g.

```
OutOfRangeException oe =
                new OutOfRangeException("Out of range");
throw oe;
```

OR

```
throw new OutOfRangeException("Out of range");
```

Checked and Unchecked Exceptions

- A checked exception must either be caught by a method or it must be listed in the throws clause of any method that may throw or propagate it. Otherwise compiler complains.

```
public void test( int j )
{
    try {
        if( j < 1 || j > 15 )
            throw new OutOfRangeException("Out of range");
        System.out.println( j + " is in range." );
    }
    catch( OutOfRangeException e ) { …… }
}
```

Checked and Unchecked Exceptions

- **A checked exception must either be caught by a method or it must be listed in the throws clause of any method that may throw or propagate it. Otherwise compiler complains.**

```
public void test( int j ) throws OutOfRangeException
{
    if( j < 1 || j > 15 )
        throw new OutOfRangeException("Out of range");
    System.out.println( j + " is in range." );
}
```

Checked and Unchecked Exceptions

```
public void testRange( int k )
{
    try {
        test( k );
    }
    catch( OutOfRangeException e ) { }
}

public void test( int j ) throws OutOfRangeException
{
    if( j < 1 || j > 15 )
        throw new OutOfRangeException("Out of range");
    System.out.println( j + " is in range." );
}
```

Checked and Unchecked Exceptions

```
public void testRange( int j ) throws OutOfRangeException
{
    test( j );
}

public void test( int j ) throws OutOfRangeException
{
    if( j < 1 || j > 15 )
        throw new OutOfRangeException("Out of range");
    System.out.println( j + " is in range." );
}
```

Checked and Unchecked Exceptions

- **An unchecked exception requires no throws clause in the method header.**

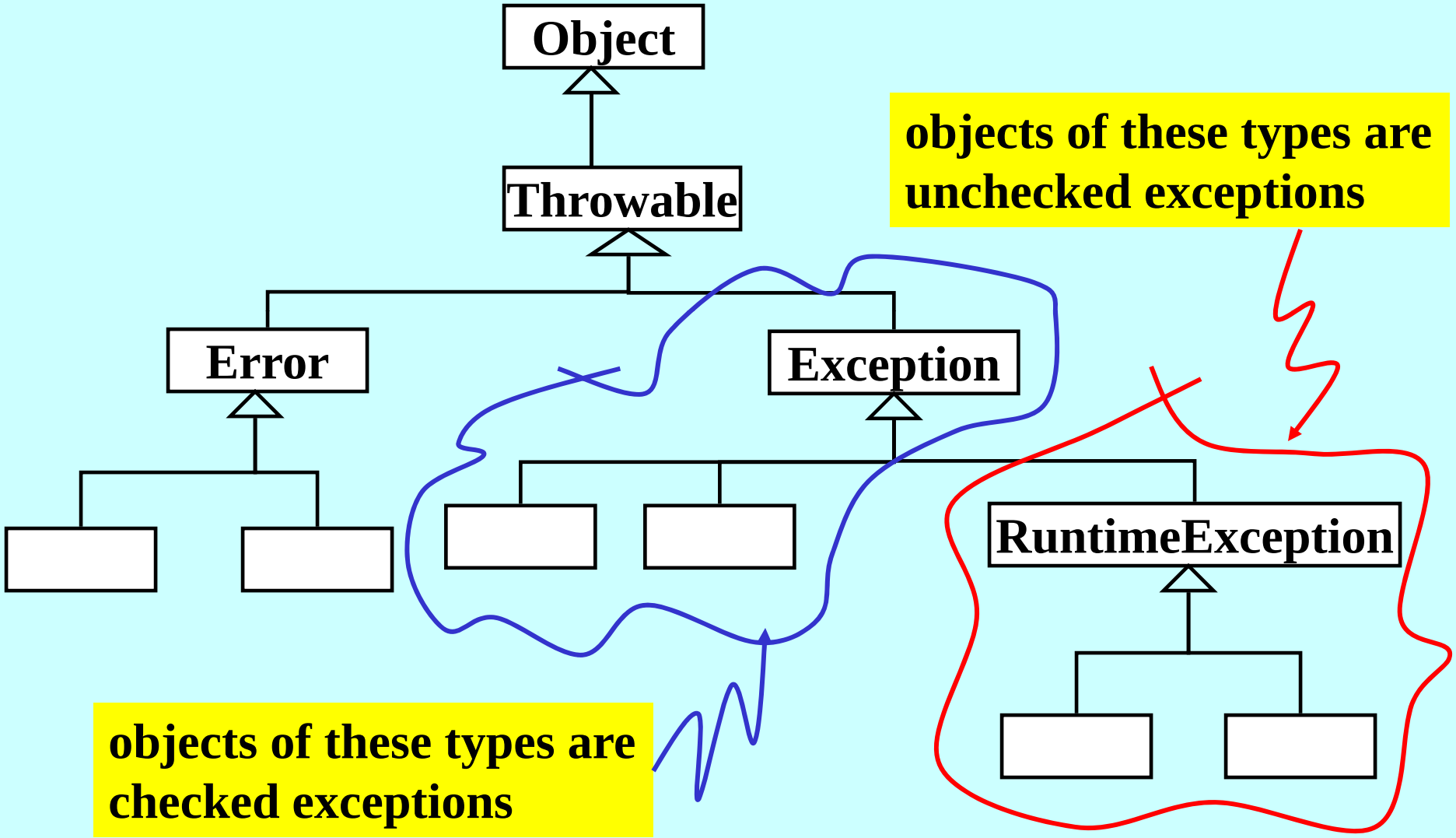
- **Unchecked exceptions: objects of type RuntimeException class or any of its descendants. e.g. ArithemticException is an unchecked exception**

- **All other exceptions are checked exceptions.**

  e.g. OutOfRangeException is a checked exception.

  Why ??

Checked and Unchecked Exceptions



Example

Output
for user input 21 ?
(see next slide)

```
public class CreatingExceptions
{
    public static void
        main(String[] args) throws OutOfRangeException
    {
        final int MIN = 1, MAX = 20;
        Scanner scan = Scanner.create(System.in);
        int j = scan.nextInt();

        if( j < MIN || j > MAX )
            throw new OutOfRangeException();

        System.out.println( j + " is in range");
    }
}
```

compile ??

Output:

```
Exception in thread "main" OutOfRangeException
        at CreatingExceptions.main(CreatingExceptions.java:13)
```

Throwing Multiple Exceptions

```
public void test( int j )
                throws IOException, OutOfRangeException
{
    ……
}
```

Throwing Exceptions

```
public void test( int j ) throws Exception
{
    ……
}
```

This compiles since IOException and OutOfRangeException are both descendants of Exception class

Throwing Exceptions

```
public void testMe( int j )
                throws IOException, OutOfRangeException
{
    ……
}

public void foo( int k )
{
    try {
        testMe( k );
    }
    catch( Exception e ) { …… }
}
```