Exceptions

(http://docs.oracle.com/javase/tutorial/essential/exceptions/index.html)

Objectives

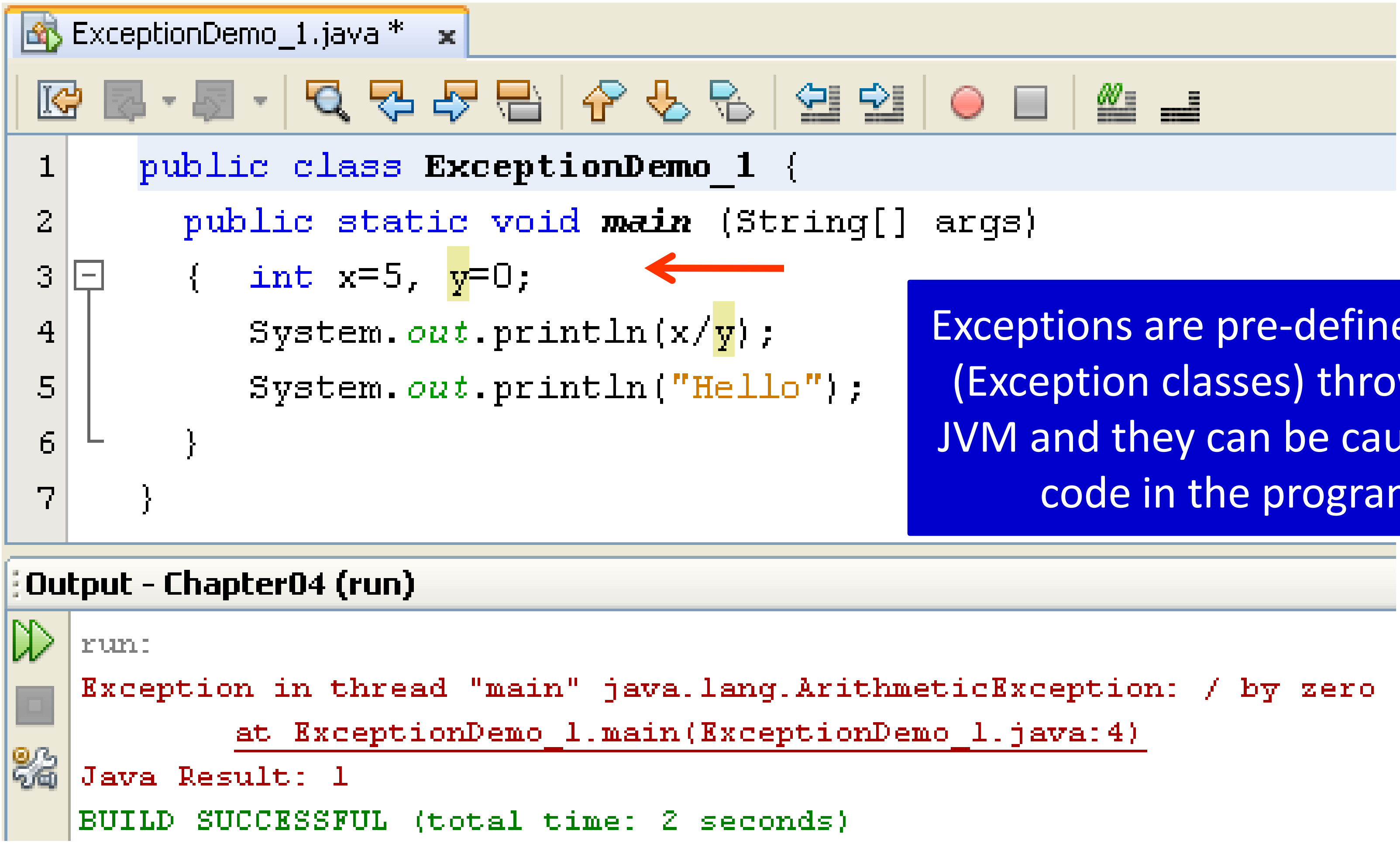
- Exception Handling
  - try block
  - catch block
  - finally block
  - custom exception class

Exceptions

- **Exception**: Error beyond the control of a program. When an exception occurs, the program will terminate abruptly.
- When a program is executing something occurs that is not quite normal from the point of view of the goal at hand.
- For example:
  - a user might type an invalid filename;
  - An accessed file does not exist of might contain corrupted data;
  - a network link could fail;
  - …
- Circumstances of this type are called *exception conditions* in Java and are represented using objects (All exceptions descend from the java.lang.**Throwable**).

Exceptions

- The following program causes an exception.



ExceptionDemo_1.java *

```
public class ExceptionDemo_1 {
  public static void main (String[] args)
  {   int x=5, y=0;
      System.out.println(x/y);
      System.out.println("Hello");
  }
}
```

Output - Chapter04 (run)

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionDemo_1.main(ExceptionDemo_1.java:4)
Java Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)
```

Exceptions are pre-defined data (Exception classes) thrown by JVM and they can be caught by code in the program

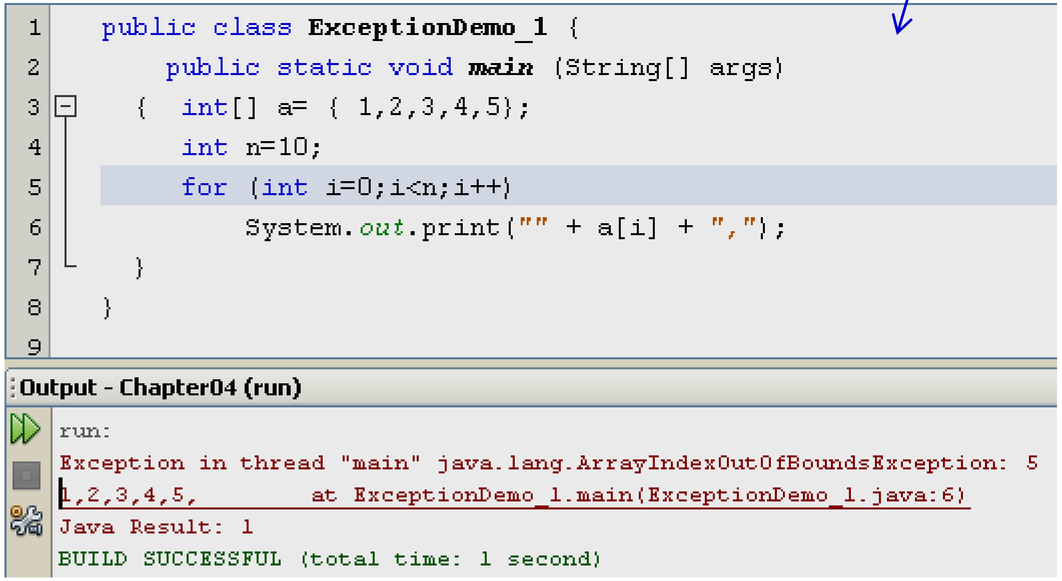
Kinds of Exceptions

- java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Error
    - java.lang.Exception
        - java.lang.RuntimeException

Checked Exceptions
(We must use the try catch blocks or throw)

Unchecked- Exceptions
Program Bugs
(We may not use the try catch blocks)

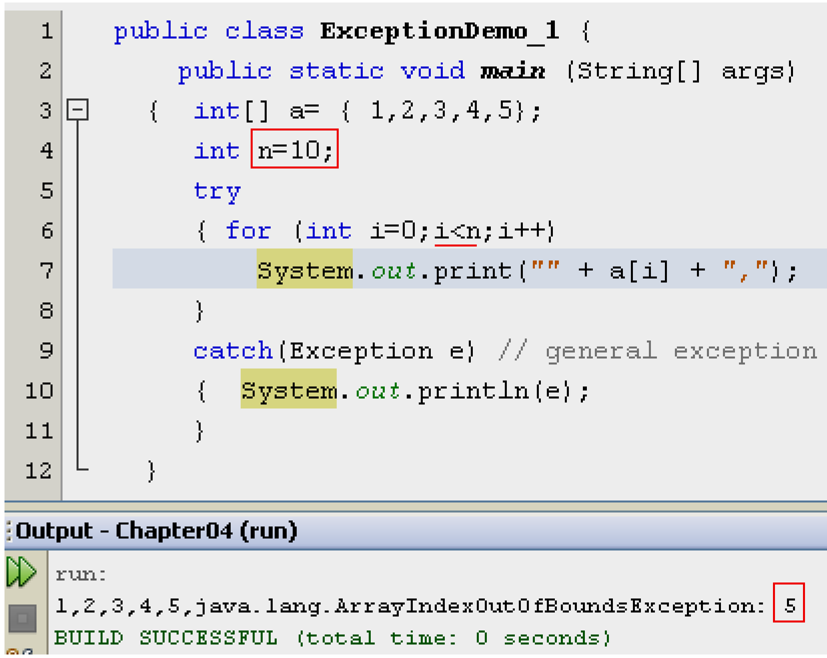
Refer to the Java.lang documentation for more information.



```
public class ExceptionDemo_1 {
    public static void main (String[] args)
  {  int[] a= { 1,2,3,4,5};
     int n=10;
     for (int i=0;i<n;i++)
         System.out.print("" + a[i] + ",");
  }
}
```

Output - Chapter04 (run)

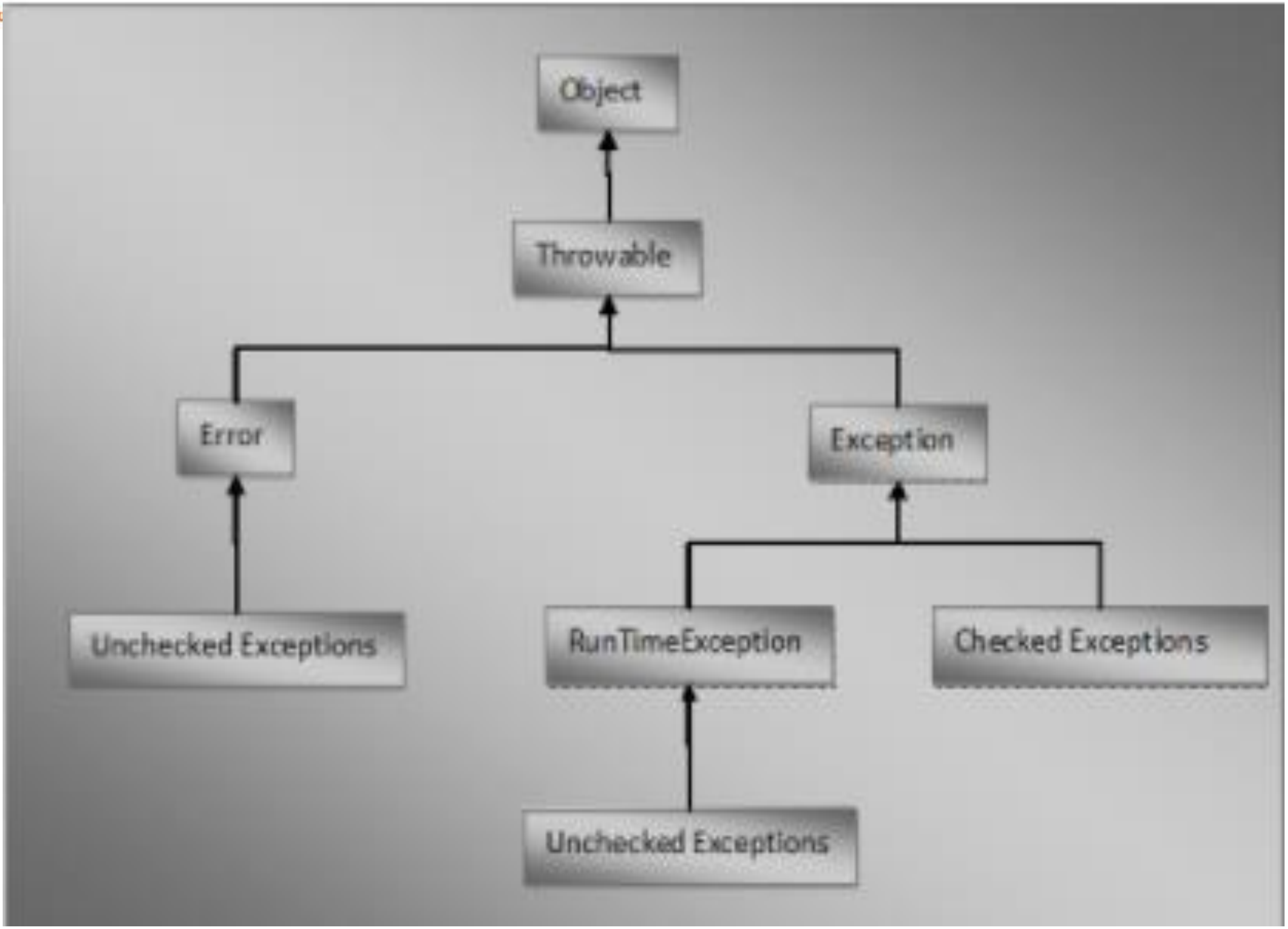
```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
1,2,3,4,5,        at ExceptionDemo_1.main(ExceptionDemo_1.java:6)
Java Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```



```
public class ExceptionDemo_1 {
    public static void main (String[] args)
  {  int[] a= { 1,2,3,4,5};
     int n=10;
     try
     { for (int i=0;i<n;i++)
          System.out.print("" + a[i] + ",");
     }
     catch(Exception e) // general exception
     {  System.out.println(e);
     }
  }
```

Output - Chapter04 (run)

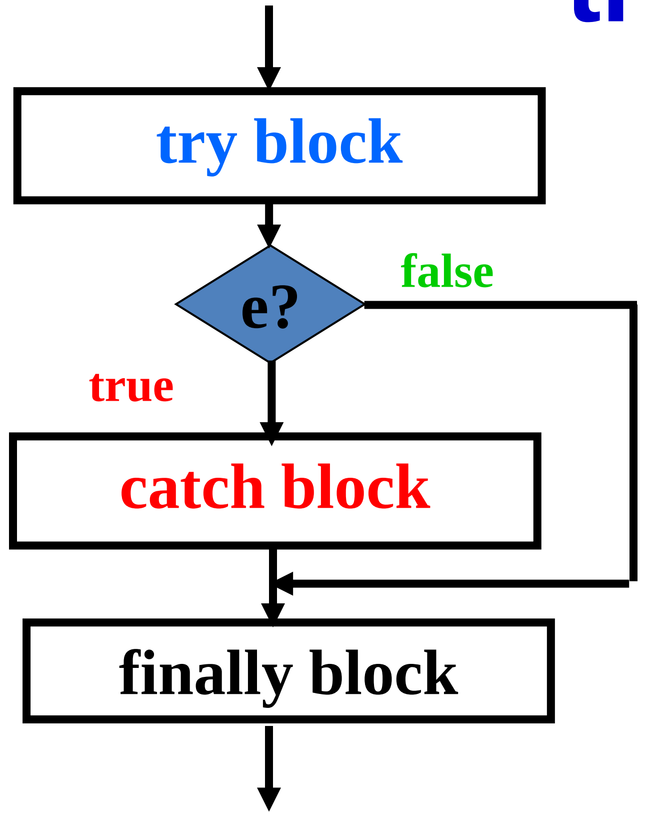
```
run:
1,2,3,4,5,java.lang.ArrayIndexOutOfBoundsException: 5
BUILD SUCCESSFUL (total time: 0 seconds)
```


Object
Throwable
Error
Exception
Unchecked Exceptions
RunTimeException
Checked Exceptions
Unchecked Exceptions

Two Kinds of Exception

- *Checked exception*
  - Must be handled by either the try-catch mechanism or the throws-declaration mechanism.
- Runtime exception
  - The right time to deal with runtime exceptions is when you’re designing, developing, and debugging your code. Since runtime exceptions <u>should never be thrown in finished code.</u>

Catching exceptions: try catch finally

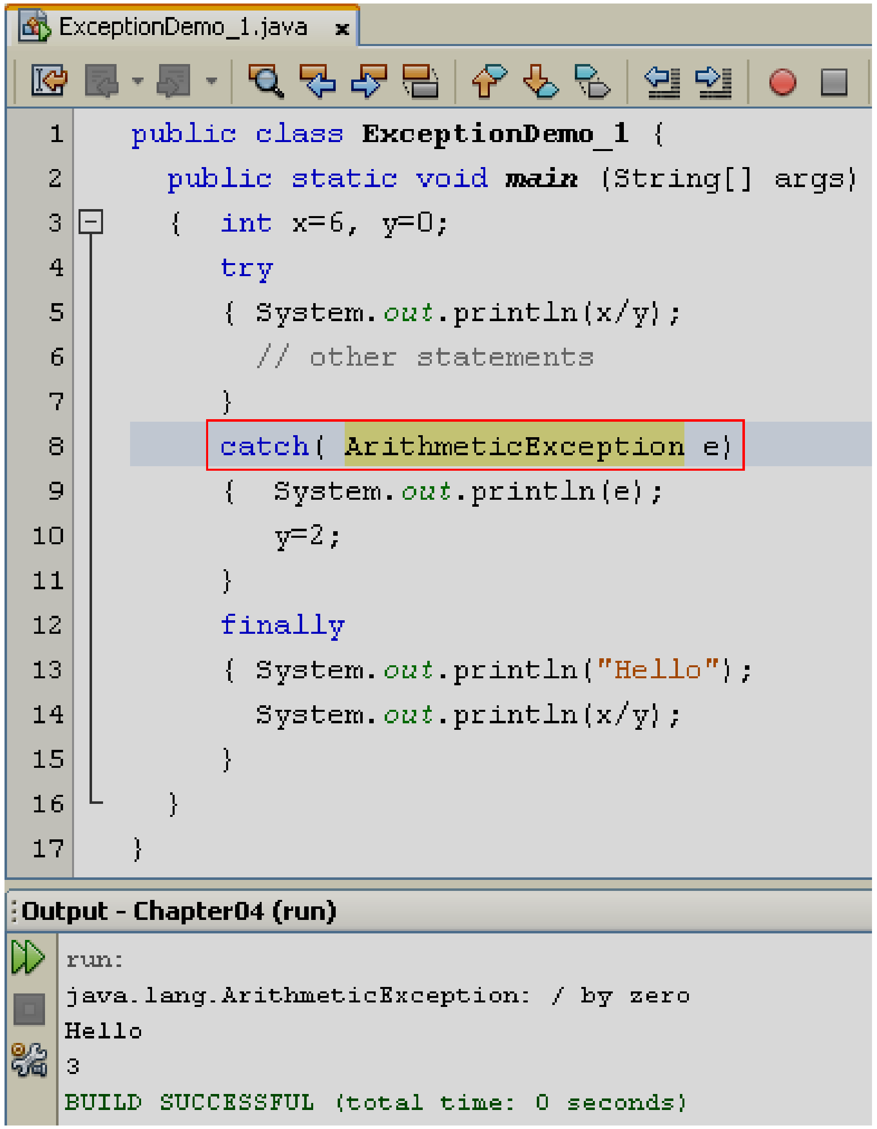


```
try {
    < statements may cause exceptions >
}
catch  (  ExceptionType1  e1 ) {
    < statements handle the situation  1>
}
catch  (  ExceptionType2  e2) {
    < statements handle the situation  2>
}
finally {
    < statements are always executed >
}
```

If no exception is thrown in the try block, all catch blocks are bypassed

If an exception arises, the first matching catch block, if any, is executed, and the others are skipped

Catching specific/general-level exception

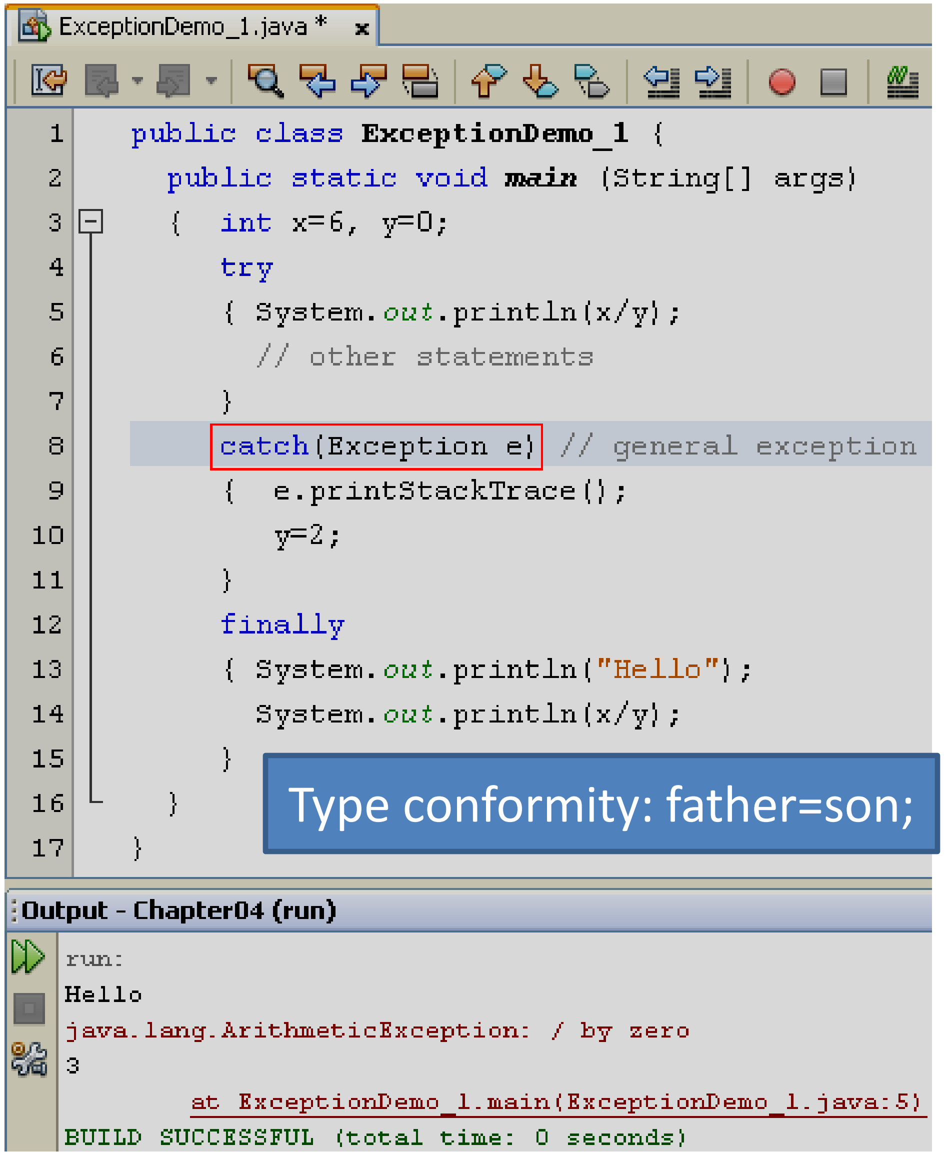


ExceptionDemo_1.java

```java
public class ExceptionDemo_1 {
  public static void main (String[] args)
  {   int x=6, y=0;
      try
      { System.out.println(x/y);
        // other statements
      }
      catch( ArithmeticException e)
      {  System.out.println(e);
         y=2;
      }
      finally
      { System.out.println("Hello");
        System.out.println(x/y);
      }
  }
}
```

Output - Chapter04 (run)

```
run:
java.lang.ArithmeticException: / by zero
Hello
3
BUILD SUCCESSFUL (total time: 0 seconds)
```



ExceptionDemo_1.java *

```java
public class ExceptionDemo_1 {
  public static void main (String[] args)
  {   int x=6, y=0;
      try
      { System.out.println(x/y);
        // other statements
      }
      catch(Exception e)  // general exception
      {  e.printStackTrace();
         y=2;
      }
      finally
      { System.out.println("Hello");
        System.out.println(x/y);
      }
  }
}
```

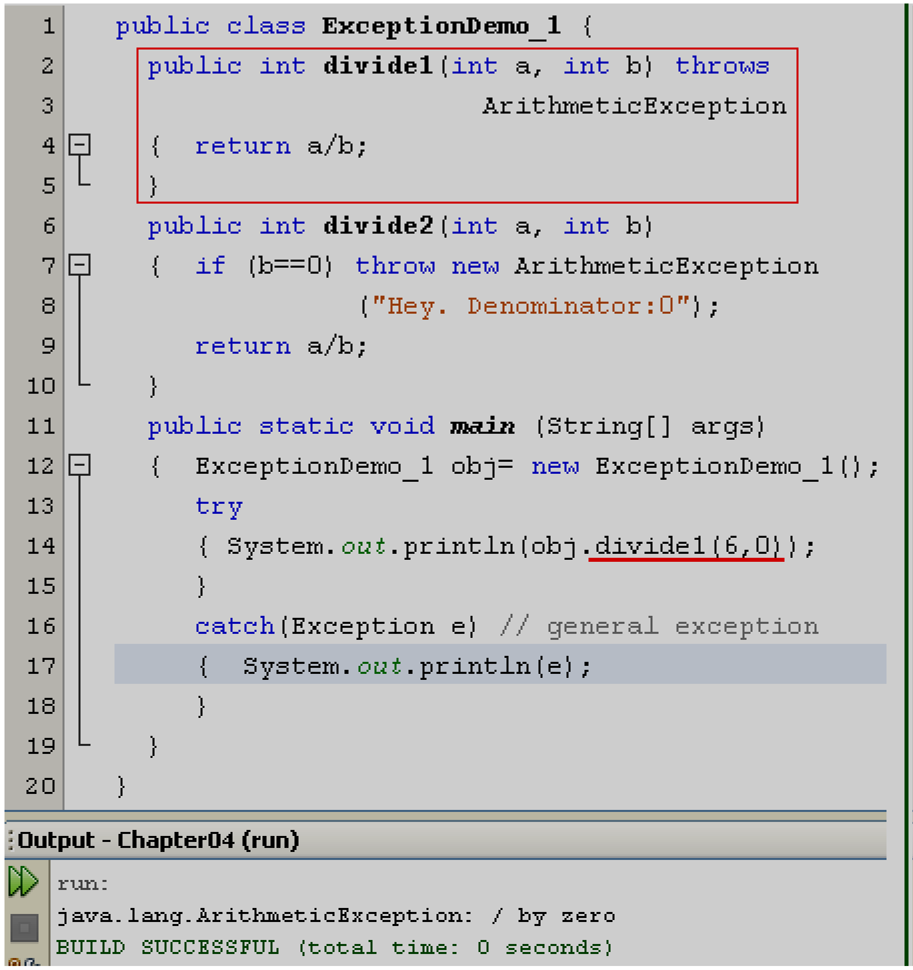
Type conformity: father=son;

Output - Chapter04 (run)

```
run:
Hello
java.lang.ArithmeticException: / by zero
3
        at ExceptionDemo_1.main(ExceptionDemo_1.java:5)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Throwing exceptions in methods

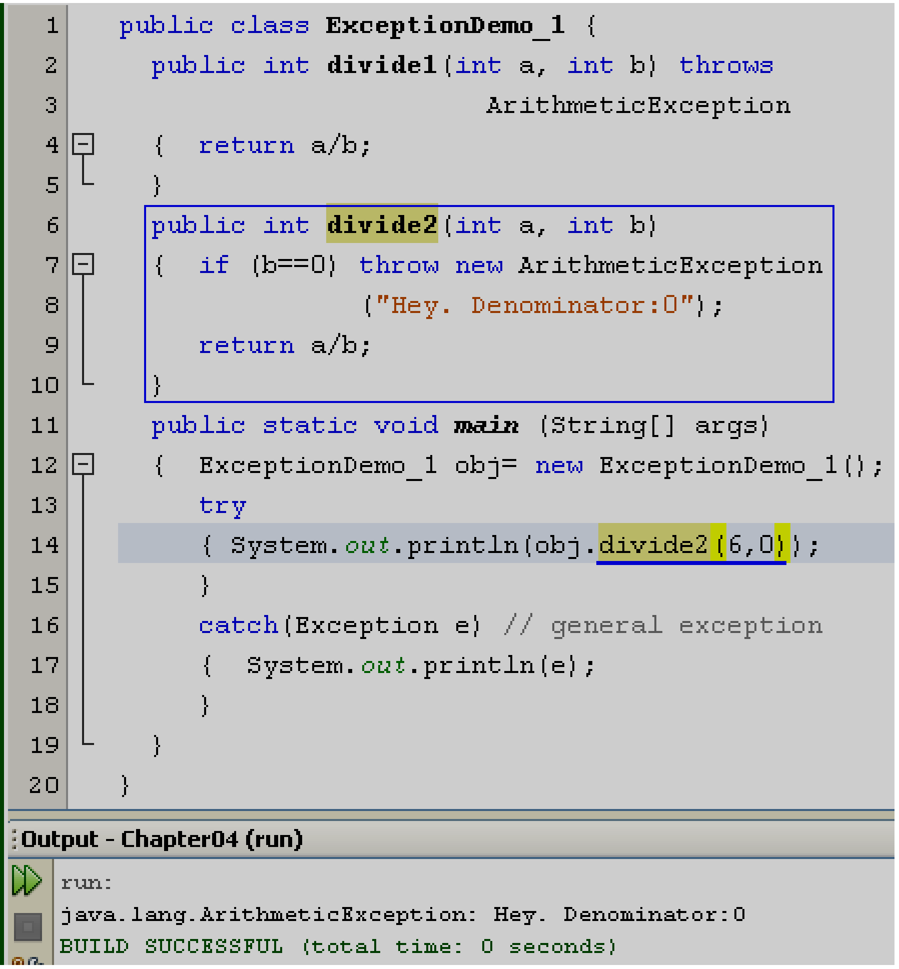
May we intentionally throw an exception? → YES



```
public class ExceptionDemo_1 {
   public int divide1(int a, int b)  throws
                                   ArithmeticException
   {  return a/b;
   }
   public int divide2(int a, int b)
   {  if (b==0) throw new ArithmeticException
                   ("Hey. Denominator:0");
      return a/b;
   }
   public static void main (String[] args)
   {  ExceptionDemo_1 obj= new ExceptionDemo_1();
      try
      { System.out.println(obj.divide1(6,0));
      }
      catch(Exception e) // general exception
      {  System.out.println(e);
      }
   }
}
```

Output - Chapter04 (run)

```
run:
java.lang.ArithmeticException: / by zero
BUILD SUCCESSFUL (total time: 0 seconds)
```

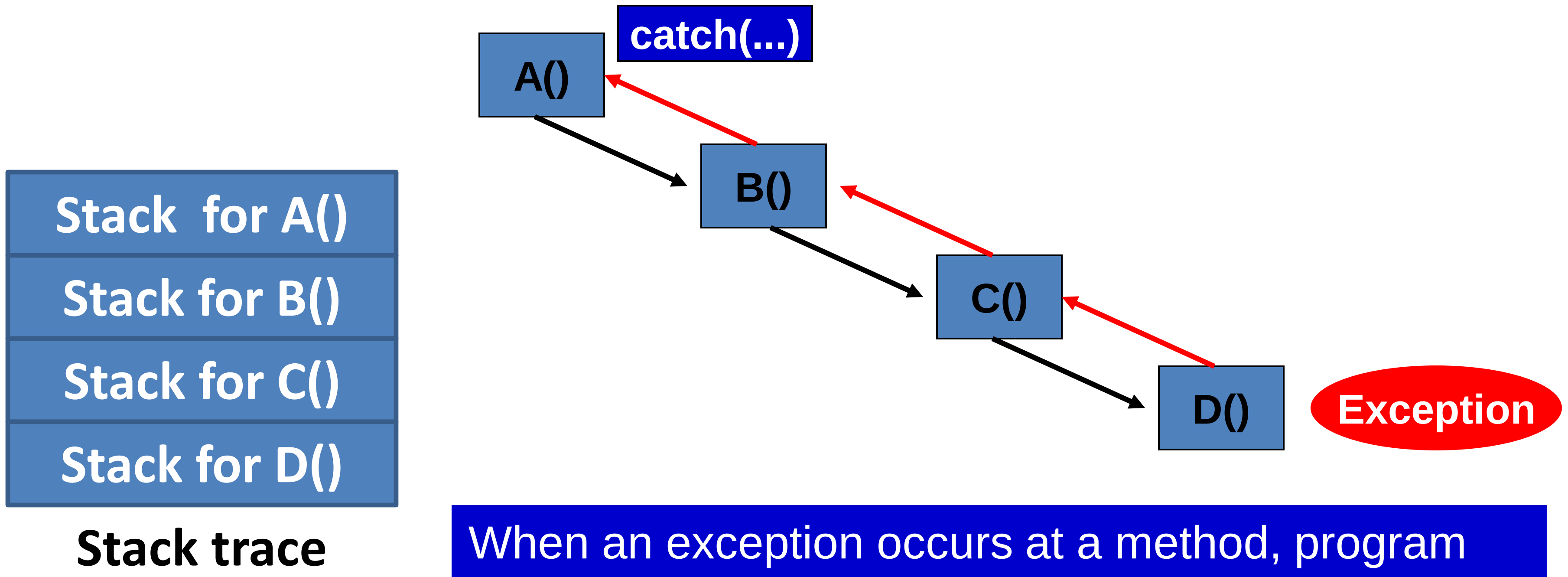


```
public class ExceptionDemo_1 {
   public int divide1(int a, int b)  throws
                                   ArithmeticException
   {  return a/b;
   }
   public int divide2(int a, int b)
   {  if (b==0) throw new ArithmeticException
                   ("Hey. Denominator:0");
      return a/b;
   }
   public static void main (String[] args)
   {  ExceptionDemo_1 obj= new ExceptionDemo_1();
      try
      { System.out.println(obj.divide2(6,0));
      }
      catch(Exception e) // general exception
      {  System.out.println(e);
      }
   }
}
```

Output - Chapter04 (run)

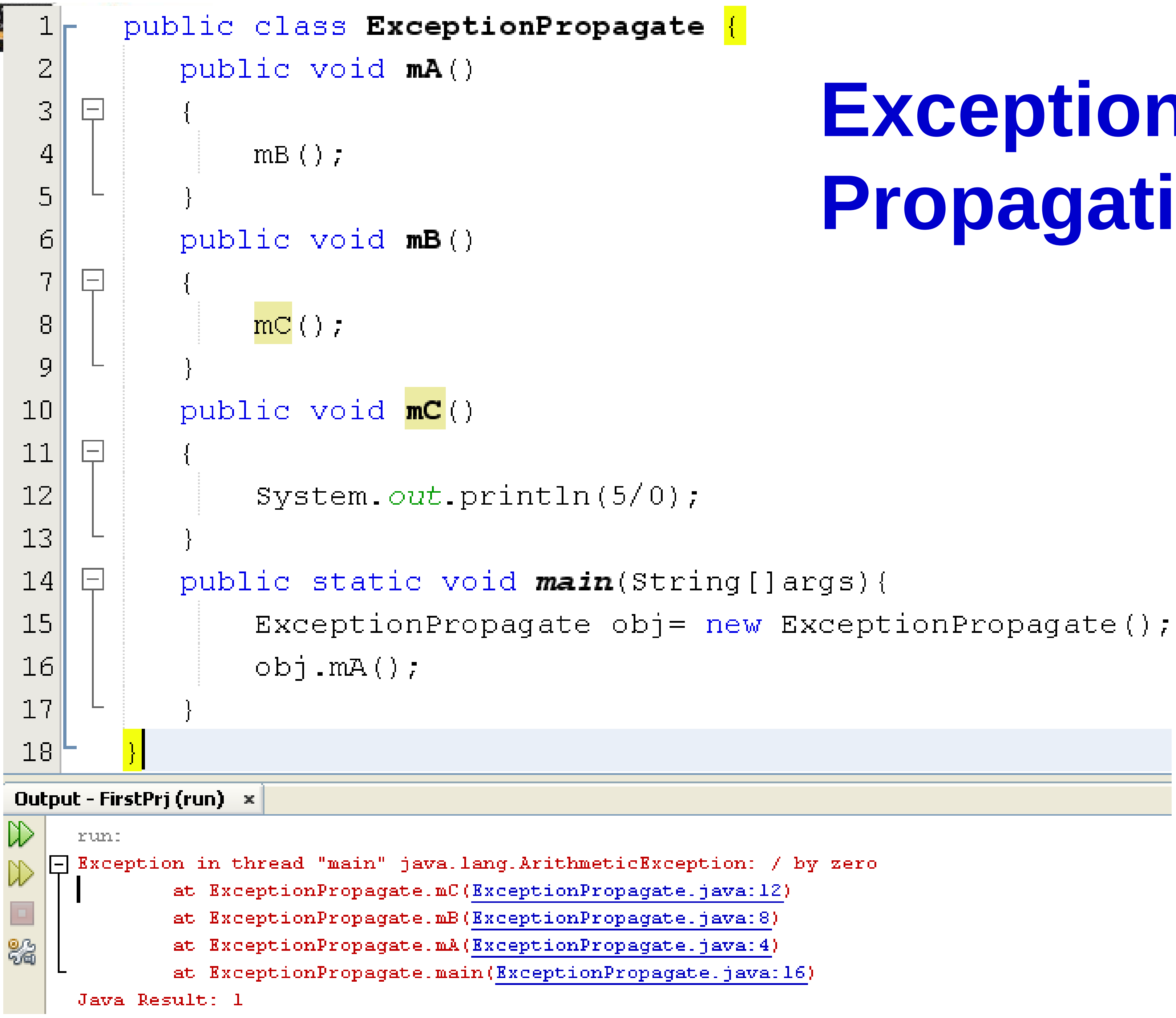
```
run:
java.lang.ArithmeticException: Hey. Denominator:0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Exception Propagations



When an exception occurs at a method, program stack is containing running methods (method A calls method B,….). So, we can trace statements related to this exception.

Exception Propagations



```
public class ExceptionPropagate {
    public void mA()
    {
        mB();
    }
    public void mB()
    {
        mC();
    }
    public void mC()
    {
        System.out.println(5/0);
    }
    public static void main(String[]args){
        ExceptionPropagate obj= new ExceptionPropagate();
        obj.mA();
    }
}
```

Output - FirstPrj (run)

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionPropagate.mC(ExceptionPropagate.java:12)
        at ExceptionPropagate.mB(ExceptionPropagate.java:8)
        at ExceptionPropagate.mA(ExceptionPropagate.java:4)
        at ExceptionPropagate.main(ExceptionPropagate.java:16)
Java Result: 1
```

Catching Exceptions…

Using try…catch to input an integer   10<=n<=50

```
Scanner in = new Scanner(System.in);
boolean cont = true;
int n;
do {
   try {
          System.out.print(“Enter a whole number: ");
          a = Integer.parseInt(in.nextLine());
          cont = false;
} catch (Exception e) {
          System.out.println("Required integer!");
     }
} while (cont == true|| n<10 || n>50);
```

The *finally* block (1)

- A try block may optionally have a finally block associated with it.
- The code within a finally block is *guaranteed* to execute no matter what happens in the try/catch code that precedes it.
    - The try block executes to completion without throwing any exceptions whatsoever.
    - The try block throws an exception that is handled by one of the catch blocks.
    - The try block throws an exception that is ***not* handled by *any* of the catch blocks**

Nesting of try/catch Blocks

- A try statement may be nested inside either the try or catch block of another try statement.

```
try {
   // Pseudo code.
   open a user-specified file
   }
   catch (FileNotFoundException e) {
      try {
          // Pseudo code.
          open a DEFAULT file instead ...
      }
      catch (FileNotFoundException e2) {
         // Pseudo code.
         attempt to recover ...
      }
}
```

Creating Your Own Exception Classes (1)

- Decide whether you want a checked or a runtime exception.
  - Checked exceptions should extend java.lang.Exception or one of its subclasses.
  - Runtime exceptions should extend java.lang.RuntimeException or one of its subclasses

Creating Your Own Exception Classes (2)

Create your own exception class with it’s constructor

```
class InvalidAge extends Exception{
    public InvalidAge(String mes) {
        super(mes);
    }
}
```

Creating Your Own Exception Classes (3)

```
//Use it in some method
class MyClass{
    public void MyMethod(int a) throws InvalidAge{
        if(a<0)
            throw new InvalidAge("Age invalid!");
    }
}
```

Creating Your Own Exception Classes (4)

```
//Using try-catch when this method is called
try {
        MyClass class1 = new MyClass();
        class1.MyMethod(-5);
  } catch (InvalidAge ex) {
        System.out.println(ex.getMessage());
  }
```

Exceptions and Overriding

- When you extend a class and override a method, the Java compiler insists (đòi hỏi) that all exception classes thrown by the new method must be the same as, or subclasses of, the exception classes thrown by the original method.

STARS RATED FOR EXCELLENCE 2012 · TRƯỜNG ĐẠI HỌC FPT

Assertions

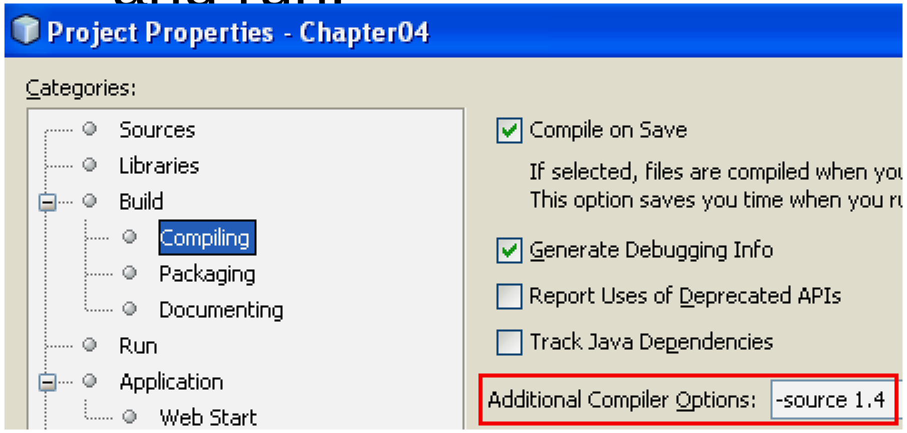
- Assertions are introduced in Java 1.4
- 2 Ways of writing assertion statements:

  `assert` expression; // true-false condition

  `assert` *expression1:expression2;* //

  condiontion:ExceptionMessage

- You must specify options when the program is compiled and run.


Project Properties - Chapter04

Categories:

- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Documenting
- Run
- Application
  - Web Start

[✓] Compile on Save

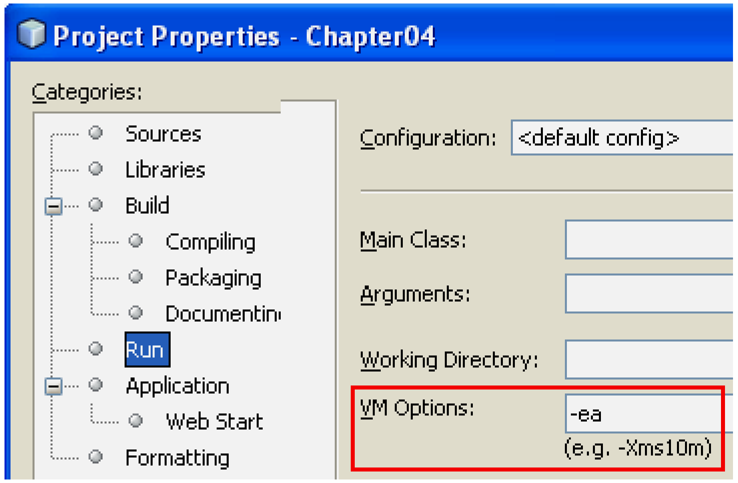
If selected, files are compiled when you
This option saves you time when you r

[✓] Generate Debugging Info

[ ] Report Uses of Deprecated APIs

[ ] Track Java Dependencies

Additional Compiler Options: -source 1.4


Project Properties - Chapter04

Categories:

- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Documentin
- Run
- Application
  - Web Start
- Formatting

Configuration: <default config>

Main Class:

Arguments:

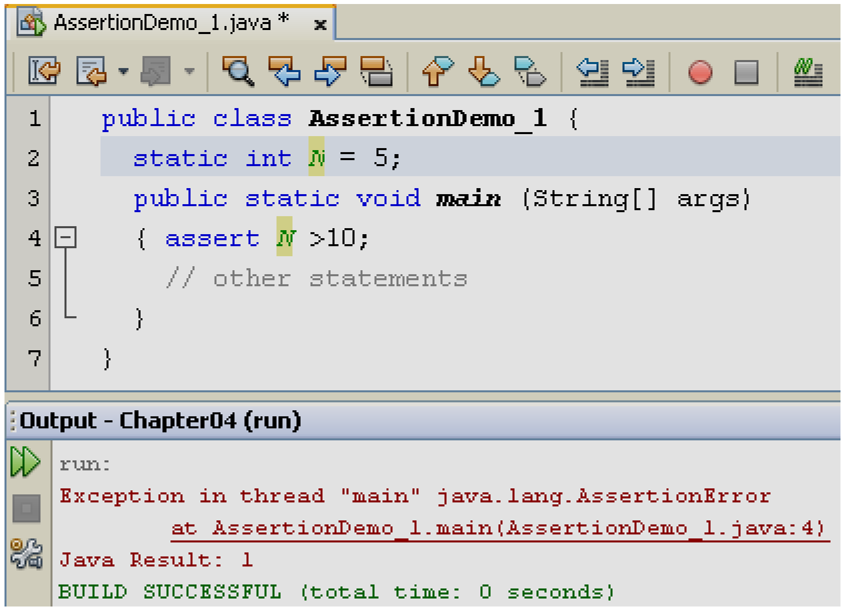
Working Directory:

VM Options: -ea

(e.g. -Xms10m)

We can replace an assertion with an ***if*** statement.
In Java from 1.5, the keyword ***assert*** is removed.

Assertions…

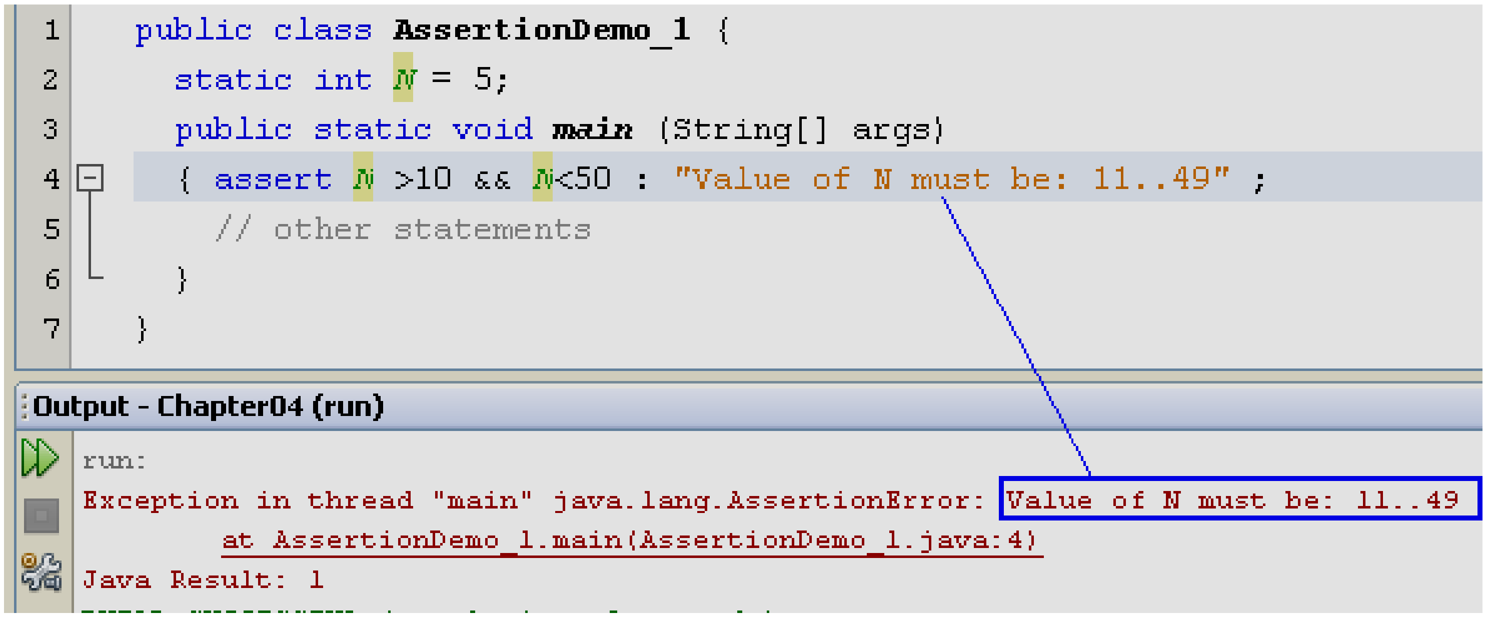


AssertionDemo_1.java *

```java
public class AssertionDemo_1 {
  static int N = 5;
  public static void main (String[] args)
  { assert N >10;
    // other statements
  }
}
```

Output - Chapter04 (run)

```
run:
Exception in thread "main" java.lang.AssertionError
        at AssertionDemo_1.main(AssertionDemo_1.java:4)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```



```java
public class AssertionDemo_1 {
  static int N = 5;
  public static void main (String[] args)
  { assert N >10 && N<50 : "Value of N must be: 11..49" ;
    // other statements
  }
}
```

Output - Chapter04 (run)

```
run:
Exception in thread "main" java.lang.AssertionError: Value of N must be: 11..49
        at AssertionDemo_1.main(AssertionDemo_1.java:4)
Java Result: 1
```

Summary

- Exception Handling
- Multiple Handlers
- Code Finalization and Cleaning Up (finally block)
- Custom Exception Classes
- Assertions