# JAVA PROGRAMMING BASICS

Module 2: Java Object-oriented Programming

# Training program

1. Classes and Instances
2. The Methods
3. The Constructors
4. Static elements
5. Initialization sections
6. Package
7. Inheritance and Polymorphism
8. Abstract classes and interfaces
9. String processing
10. Exceptions and Assertions
11. Nested classes
12. Enums
13. Wrapper classes for primitive types
14. Generics
15. Collections
16. Method overload resolution
17. Multithreads
18. Core Java Classes
19. Object Oriented Design

# Module contents

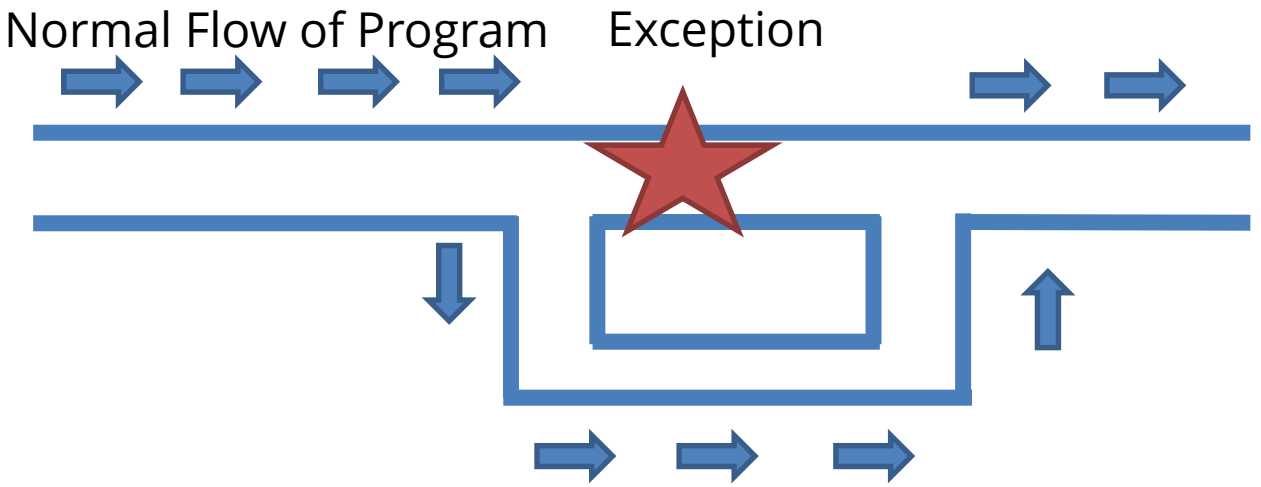
- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - Checked and Unchecked Exceptions
  - The try-catch-finally block
  - Multiple catch blocks
  - The throw and throws keywords
  - Call stack and Exception propagation
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# Module contents

- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - The try-catch-finally block
  - The throw and throws keywords
  - Multiple catch blocks
  - Call stack and Exception propagation
  - Checked and Unchecked Exceptions
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# The Exceptions.  Java Exceptions hierarchy 1/6

- Definition: An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.



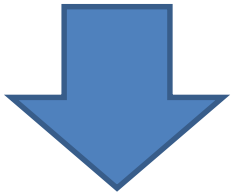

**Exception handling** Alternate way to continue flow of program

# The Exceptions.  Java Exceptions hierarchy 2/6

- Exceptions examples:
- – Divide by zero
- – Accessing the elements of an array beyond its range
- – Invalid input
- – Hard disk crash
- – Opening a non-existent file
- – Heap memory exhausted

# The Exceptions.  Java Exceptions hierarchy 3/6

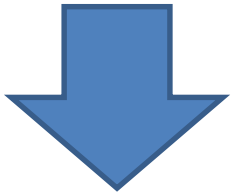
```
1. public static void main(String[] arg){
2.     int x1 = 10, x2 = 0;
3.     int y = x1/x2;
4. }
```



```
Exception in thread "main" java.lang.ArithmeticException:
/ by zero
        at
com.brainacad.oop1.testexcp.Main.main(Main.java:6)
```

# The Exceptions.  Java Exceptions hierarchy 4/6

```
1. public static void main(String[] arg){
2.     int[] arr = {1,2,3};
3.     int x = arr[4];
4. }
```



```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 4
        at
com.brainacad.oop1.testexcp.Main.main(Main.java:6)
```

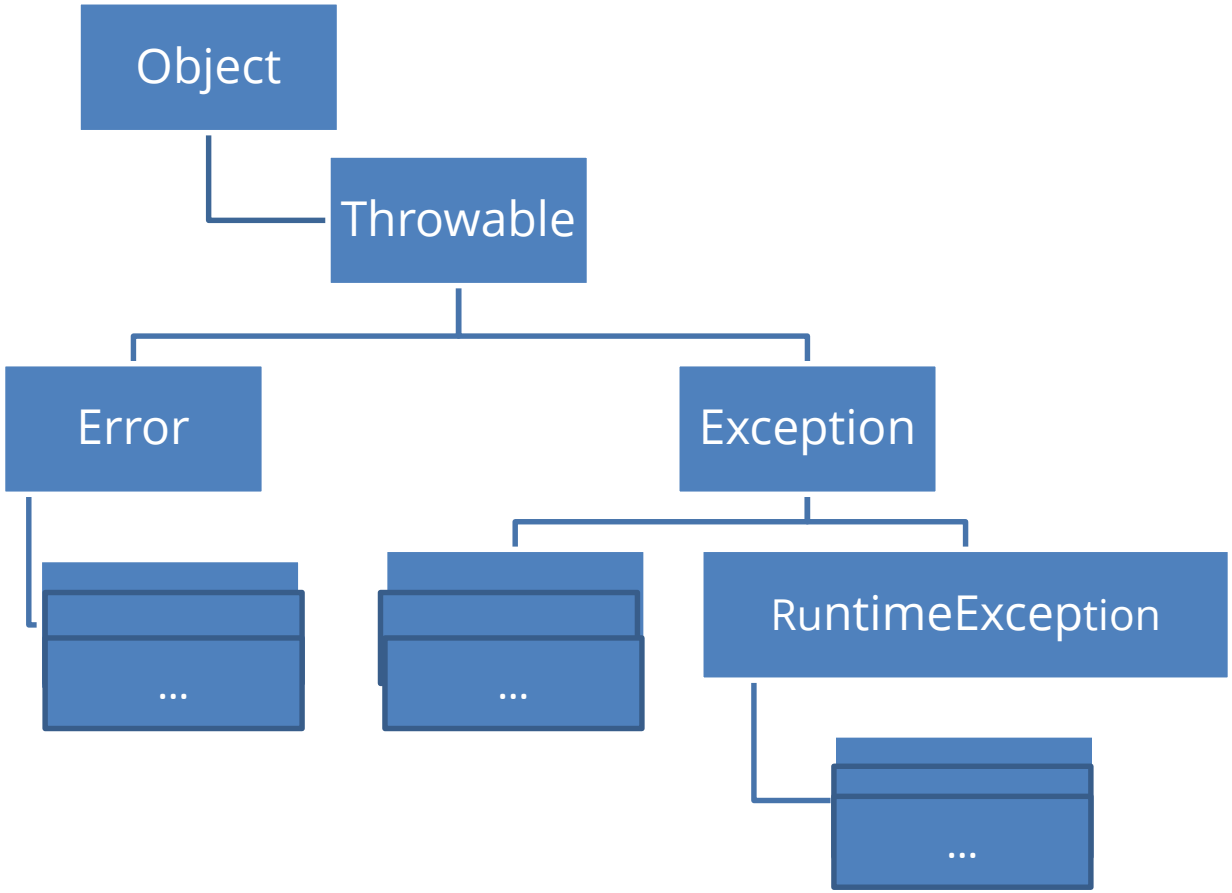
# The Exceptions.  Java Exceptions hierarchy 5/6

```
1.  public static void main(String[] arg){
2.      doJob();
3.  }
4.  public static void doJob(){
5.      doJob();
6.  }
```

```
Exception in thread "main" java.lang.StackOverflowError
        at
com.brainacad.oop1.testexcp.TestError.doJob(TestError
.java:11)
```

# The Exceptions. Java Exceptions hierarchy 6/6

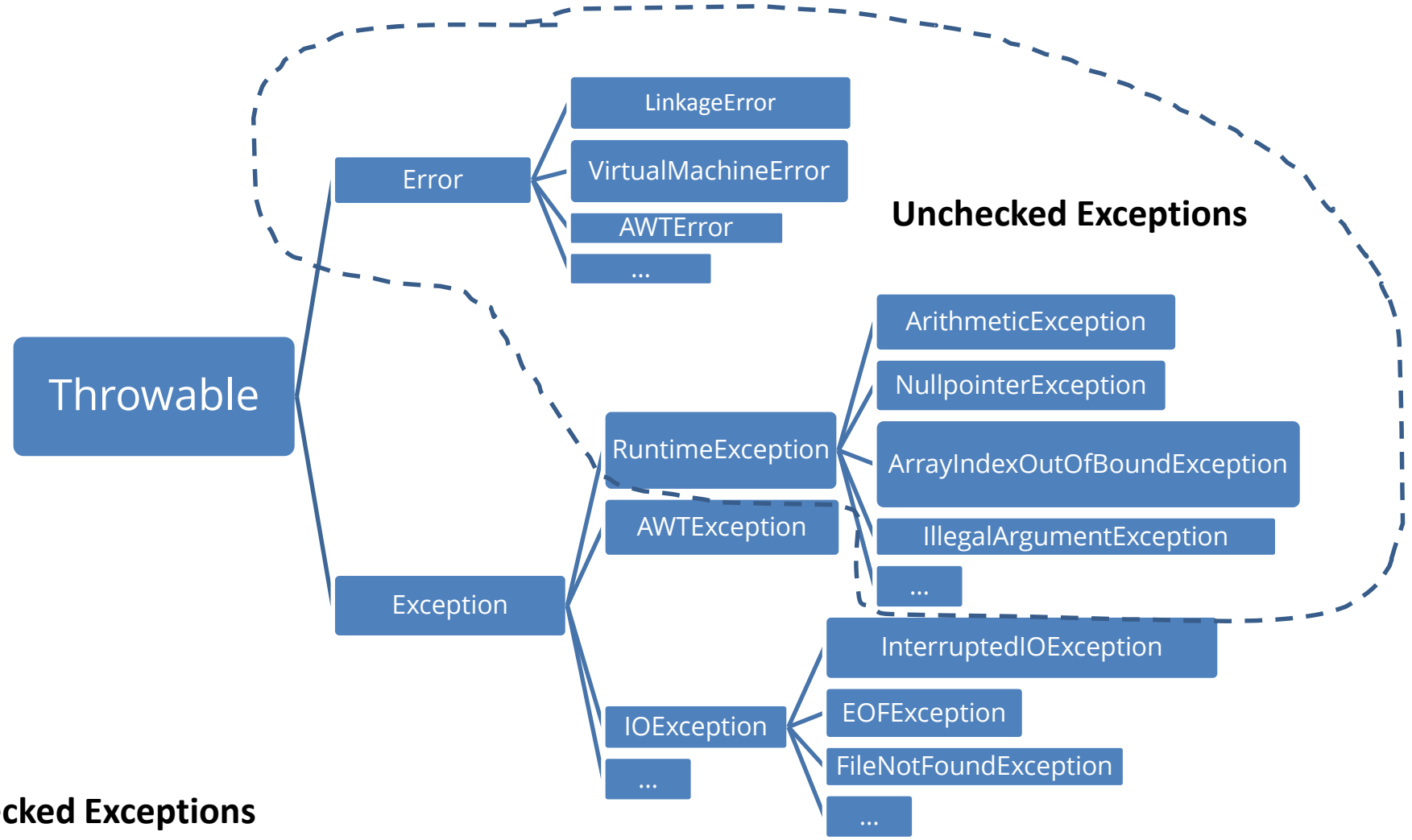
- Java Exceptions hierarchy



# Module contents

- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - Checked and Unchecked Exceptions
  - The try-catch-finally block
  - The throw and throws keywords
  - Multiple catch blocks
  - Call stack and Exception propagation
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# Checked and Unchecked Exceptions




**Checked Exceptions**
(Must be caught or declared to be thrown)

# Module contents

- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - Checked and Unchecked Exceptions
  - The try-catch-finally block
  - The throw and throws keywords
  - Multiple catch blocks
  - Call stack and Exception propagation
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# The try-catch-finally block 1/7

- In general, a try block looks like the following:

```
1. try {
2.         <code to be monitored for exceptions>
3.   } catch (<ExceptionType> <ObjName>) {
4.         <handler if ExceptionType occurs>
5.   }
```

- If an exception occurs within the try block, that exception is handled by an exception handler associated with it.
- The catch block contains code that is executed if and when the exception handler is invoked

# The try-catch-finally block 2/7

```
1. public static void main(String[] arg){
2.     int[] arr = {1,2,3};
3.     try {
4.         int x = arr[4];
5.     }catch(ArrayIndexOutOfBoundsException e){
6.         System.out.println(e);
7.     }
8.     System.out.println("Program Finish OK!");
9. }
```

# The try-catch-finally block 3/7

- The finally block contains the code for cleaning up after a try or a catch

```
1. try {
2.         <code to be monitored for exceptions>
3. } catch (<ExceptionType1> <ObjName>) {
4.         <handler if ExceptionType1 occurs>
5. } finally {
6.         <code to be executed before the try block ends>
7. }
```

# The try-catch-finally block 4/7

```
1.  public static void testExcp() {
2.      try {
3.          int x = 10 / 0;
4.      } catch (Exception ex) {
5.          System.out.println("catch");
6.          return;
7.      } finally {
8.          System.out.println("finally");
9.      }
10. }
```

**Console output:**

start
catch
finally
end

# The try-catch-finally block 5/7

```
1. public static void main(String args[]) {
2.     System.out.println("start");
3.     testExcp();
4.     System.out.println("end");
5. }
```

**Console output:**

start
catch
finally
end

# The try-catch-finally block 6/7

```
1.  public static int testExcp2() {
2.      int y = 50;
3.      try {
4.          int x = 10 / 0;
5.      } catch (Exception ex) {
6.          System.out.println("catch");
7.          return y;
8.      } finally {
9.          System.out.println("finally");
10.         y=100;
11.     }
12.     return 0;
13. }
```

**Console output:**

start
catch
finally
50
end

# The try-catch-finally block 7/7

```
1. public static void main(String args[]) {
2.     System.out.println("start");
3.     System.out.println(testExcp2());
4.     System.out.println("end");
5. }
```

**Console output:**

start
catch
finally
50
end

# Module contents

- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - The try-catch-finally block
  - The throw and throws keywords
  - Multiple catch blocks
  - Call stack and Exception propagation
  - Checked and Unchecked Exceptions
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# Multiple catch blocks 1/4

```
1. try {
2.         <code to be monitored for exceptions>
3. } catch (<ExceptionType1> <ObjName>) {
4.         <handler if ExceptionType1 occurs>
5. }
6. ...
7. } catch (<ExceptionTypeN> <ObjName>) {
8.         <handler if ExceptionTypeN occurs>
9. }
```

# Multiple catch blocks 2/4

```
1.  public static void main(String args[]) {
2.      try {
3.          int a = Integer.parseInt(args[0]);
4.          int b = Integer.parseInt(args[1]);
5.          System.out.println(a / b);
6.      } catch (ArithmeticException e) {
7.          System.out.println(e.getMessage());
8.      } catch (ArrayIndexOutOfBoundsException e) {
9.          System.out.println(e.getMessage());
10.     } catch (Exception e) {
11.         System.out.println(e.getMessage());
12.     }
13. }
```

# Multiple catch blocks 3/4

- Multiple catches should be ordered from subclass to superclass

```
1.  public static void main(String args[]) {
2.      try {
3.          int a = Integer.parseInt(args[0]);
4.          int b = Integer.parseInt(args[1]);
5.          System.out.println(a / b);
6.      } catch (Exception ex) {
7.          //...
8.      } catch (ArrayIndexOutOfBoundsException e) {
9.          //...
10.     }
11. }
```

# Multiple catch blocks 4/4

```
1.  public static void main(String args[]) {
2.      try {
3.          int a = Integer.parseInt(args[0]);
4.          int b = Integer.parseInt(args[1]);
5.          System.out.println(a / b);
6.      } catch (ArithmeticException |
7.                  ArrayIndexOutOfBoundsException e) {
8.          System.out.println(e.getMessage());
9.      } catch (Exception e) {
10.         System.out.println(e.getMessage());
11.     }
12. }
```

# Module contents

- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - The try-catch-finally block
  - Multiple catch blocks
  - The throw and throws keywords
  - Call stack and Exception propagation
  - Checked and Unchecked Exceptions
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# The throw and throws keywords 1/6

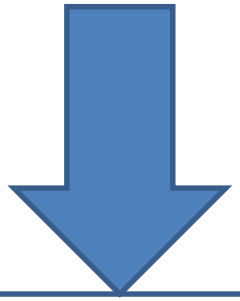
- Java allows you to throw exceptions (generate exceptions)
- **throw** <exception object>;
- An exception you throw is an object

  – You have to create an exception object in the same way you create any other object
- Example:

1. **throw new** ArithmeticException(**"test"**);

# The throw and throws keywords 2/6

```
1. public class TestThrowException {
2.     public static void main(String[] arg) {
3.         throw new RuntimeException("Test");
4.     }
5. }
```



```
Exception in thread "main" java.lang.RuntimeException: Test
        at
com.brainacad.oop1.testexcp.TestThrowException.main(TestT
hrowException.java:8)
```

# The throw and throws keywords 3/6

- A method is required to either catch or list all exceptions it might throw (except for Error or RuntimeException, or their subclasses)
- Syntax:

1. … methodName(<parameters>) **throws** <exception> {
2. //…
3. }

- If a method may cause an checked exception to occur but does not catch it, then it must say so using the **throws** keyword

# The throw and throws keywords 4/6

```
1.  public static void main(String[] arg) {
2.      try {
3.          testExcp();
4.      }catch(Exception e){
5.      System.out.println(e.getMessage());
6.      }
7.  }
8.  public static void testExcp() {
9.      throw new Exception("test");
10. }
```

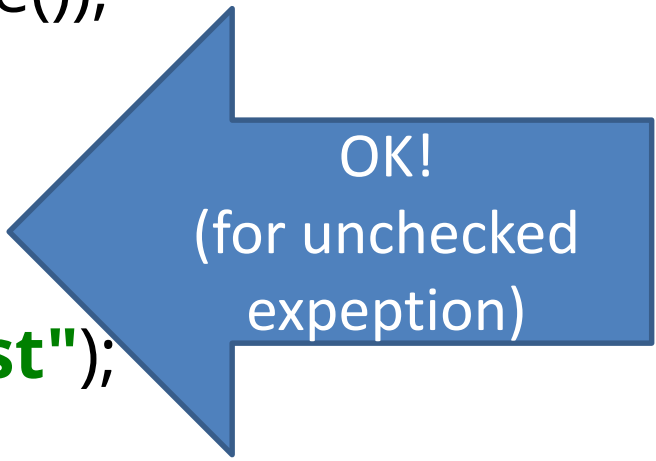
Must declare Exception using the *throws* keyword

# The throw and throws keywords 5/6

```
1.  public static void main(String[] arg) {
2.      try {
3.          testExcp();
4.      }catch(Exception e){
5.      System.out.println(e.getMessage());
6.      }
7.  }
8.  public static void testExcp() throws Exception{
9.      throw new Exception("test");
10. }
```

# The throw and throws keywords 6/6

```
1.  public static void main(String[] arg) {
2.      try {
3.          testExcp();
4.      }catch(RuntimeException e){
5.          System.out.println(e.getMessage());
6.      }
7.  }
8.  public static void testExcp(){
9.      throw new RuntimeException("test");
10. }
```

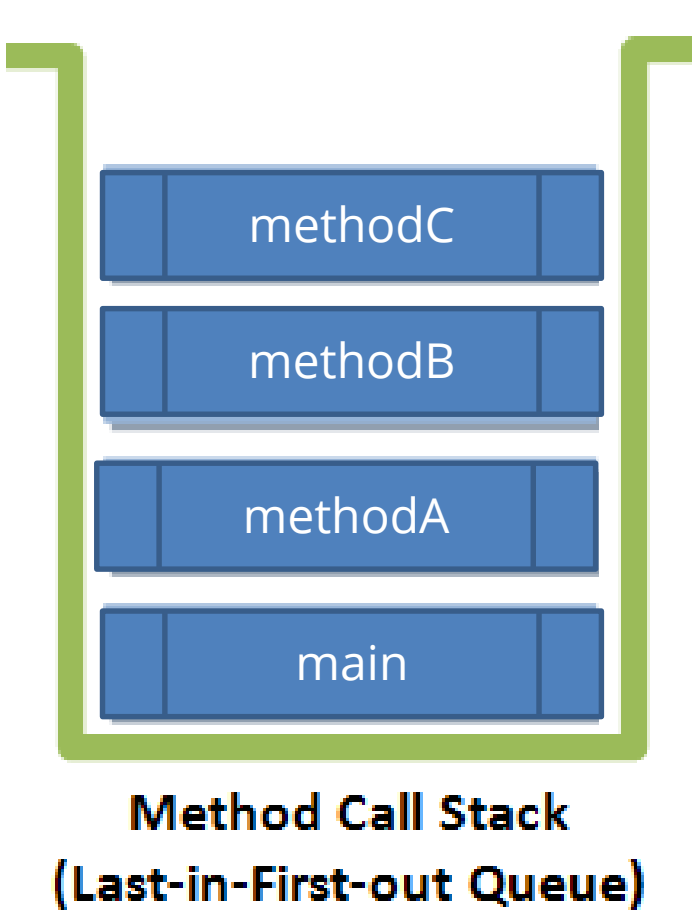


# Module contents

- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - The try-catch-finally block
  - The throw and throws keywords
  - Multiple catch blocks
  - Call stack and Exception propagation
  - Checked and Unchecked Exceptions
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

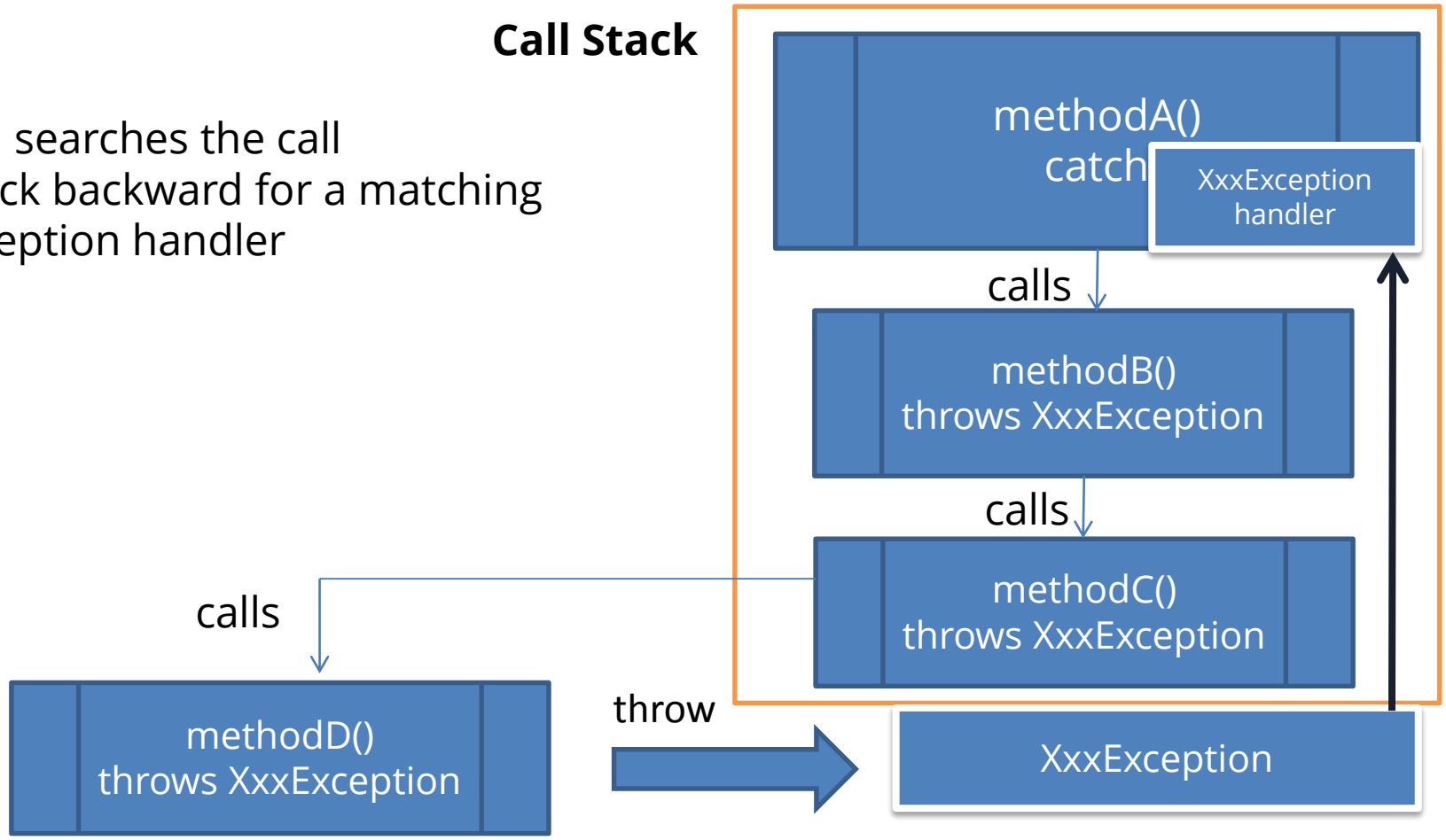
# Call stack and Exception propagation 1/4

- A typical application involve many levels of method calls which is managed by a so-called *method* ***call stack***.
- A *stack* is a ***last-in-first-ou*** *queue*.



# Call stack and Exception propagation 2/4

JVM searches the call stack backward for a matching Exception handler



# Call stack and Exception propagation 3/4

```
1.  public void doSomething() throws RuntimeException {
2.      try{
3.          doSomething2();
4.      } finally {
5.          //...
6.      }
7.  }
8.  public void doSomething2(){
9.      throw new RuntimeException();
10. }
```

# Call stack and Exception propagation 4/4

```
1.  public void doSomething() throws Exception{
2.      try{
3.          doSomething2();
4.      } catch (Exception e){
5.          throw new Exception(e.getMessage() + "more info");
6.      } finally {
7.          //...
8.      }
9.  }
10. public void doSomething2() throws Exception{
11.     throw new Exception();
12. }
```

# Module contents

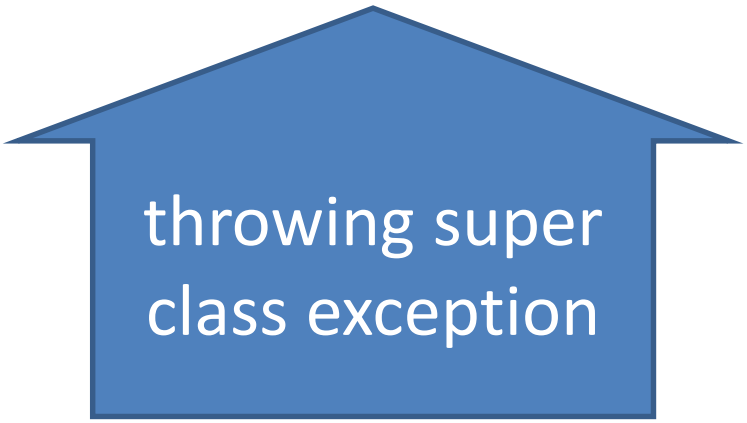
- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - Checked and Unchecked Exceptions
  - The try-catch-finally block
  - Multiple catch blocks
  - The throw and throws keywords
  - Call stack and Exception propagation
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# Rules of Exceptions in Method Overriding 1/5

- Subclass overridden method cannot throw more checked exceptions than that of super class method

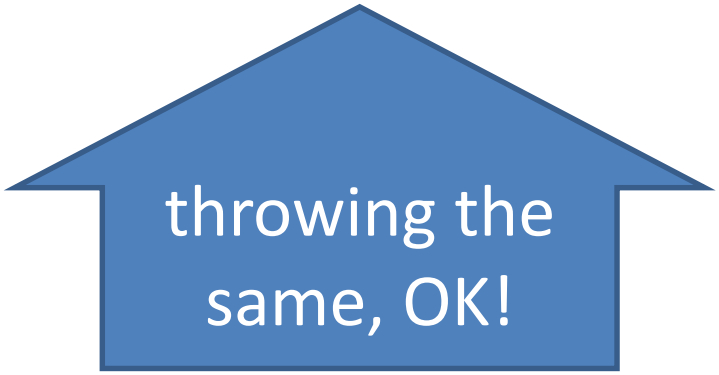
# Rules of Exceptions in Method Overriding 2/5

```
1.  class Test {
2.      public void doJob() throws IOException {
3.      //...
4.      }
5.  }
6.  class Demo extends Test {
7.      public void doJob() throws Exception {
8.      //...
9.      }
10. }
```



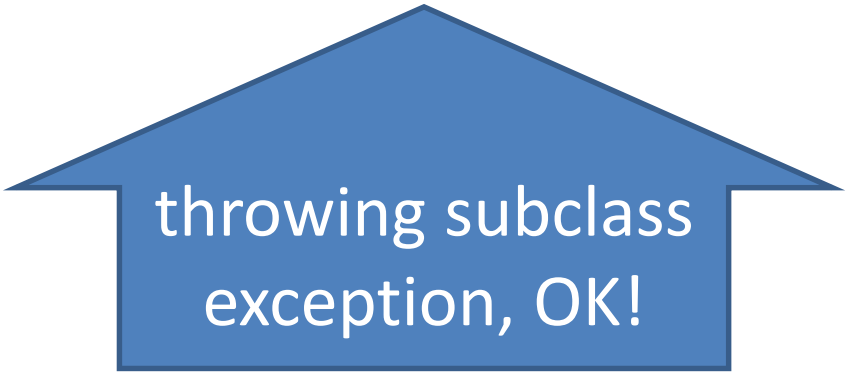
# Rules of Exceptions in Method Overriding 3/5

```
1. class Test {
2.     public void doJob() throws IOException {
3.     //...
4.     }
5. }
6. class Demo extends Test {
7.     public void doJob() throws  IOException  {
8.     //...
9.     }
```



# Rules of Exceptions in Method Overriding 4/5

```
1. class Test {
2.     public void doJob() throws IOException {
3.     //...
4.     }
5. }
6. class Demo extends Test {
7.     public void doJob() throws FileNotFoundException {
8.     //...
9.     }
```



# Rules of Exceptions in Method Overriding 5/5

```
1. class Test {
2.     public void doJob() throws IOException {
3.     //...
4.     }
5. }
6. class Demo extends Test {
7.     public void doJob() {
8.     //...
9.     }
```

# Module contents

- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - Checked and Unchecked Exceptions
  - The try-catch-finally block
  - Multiple catch blocks
  - The throw and throws keywords
  - Call stack and Exception propagation
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# Creating Exception classes 1/3

- You can create your own exceptions. All exceptions must be a child of Throwable or it subclasses.

- Creating checked Exception class

```
1. class CarTankOverflowException extends Exception{
2. }
```

- Creating unchecked Exception class

```
1. class MyException extends RuntimeException {
2. }
```

# Creating Exception classes 2/3

```
1.  public class Car {
2.      protected double gasTankMax = 40;
3.      protected double gasTank = 0;
4.      //...
5.      public void fillGasTank(double gasValue) throws
6.  CarTankOverflowException{
7.          if((gasTank+gasValue)>gasTankMax){
8.              throw new CarTankOverflowException();
9.          }
10.         gasTank+=gasValue;
11.         //...
12.     }
```

# Creating Exception classes 3/3

```
1. public static void main(String[] arg){
2.     Car myCar = new Car();
3.     try {
4.         myCar.fillGasTank(35);
5.         myCar.fillGasTank(22);
6.     }catch(Exception e){
7.         e.printStackTrace();
8.     }
9. }
```

```
com.brainacad.oop1.testcars.CarTankOverflowException
        at com.brainacad.oop1.testcars.Car.fillGasTank
```

# Module contents

- Exceptions and assertions
  - The Exceptions.  Java Exceptions hierarchy
  - Checked and Unchecked Exceptions
  - The try-catch-finally block
  - Multiple catch blocks
  - The throw and throws keywords
  - Call stack and Exception propagation
  - Rules of Exceptions in Method Overriding
  - Creating Exception classes
  - Assertions

# Assertions 1/8

- An assertion is a statement in the JavaTM programming language that enables you to test your assumptions about your program.

# Assertions 2/8

- The assertion statement has two forms.
- The first, simpler form is:

- **assert** Expression1;

- where Expression1 is a boolean expression. When the system runs the assertion, it evaluates Expression1 and if it is false throws an ***AssertionError*** with no detail message.

# Assertions 3/8

- The second form of the assertion statement is:

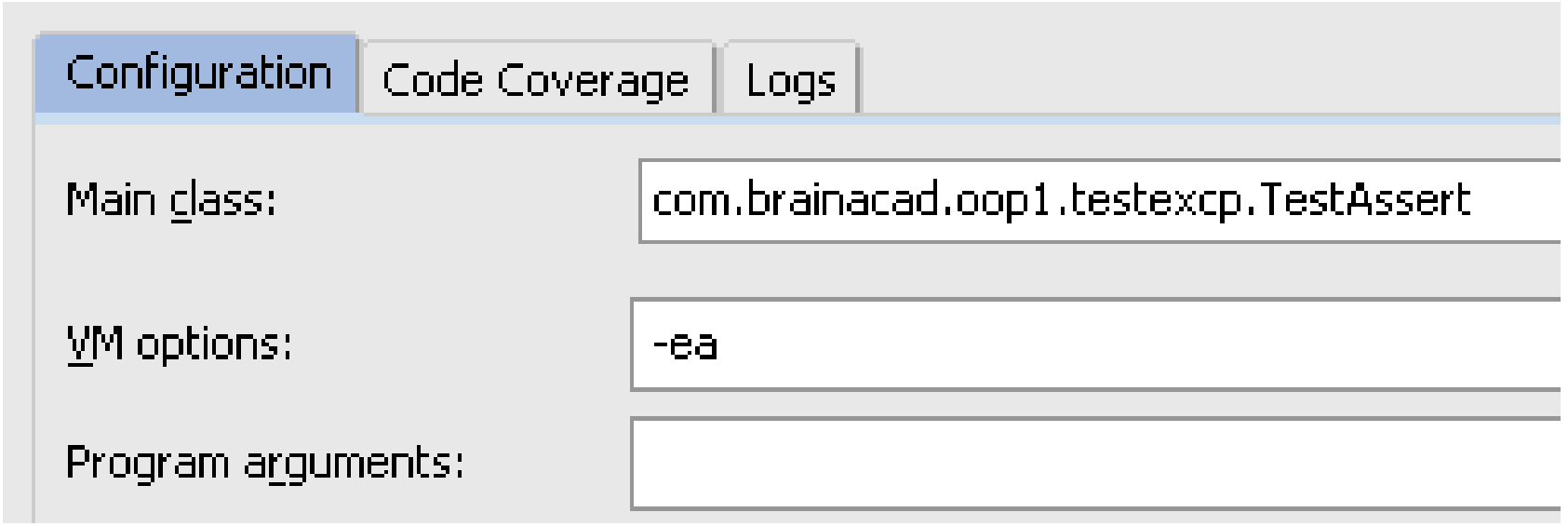
- **assert** Expression1 : Expression2 ;
- where:
- Expression1 is a boolean expression.
- Expression2 is an expression that has a value. (It cannot be an invocation of a method that is declared void.)
- The system passes the value of Expression2 to the appropriate **AssertionError** constructor.

# Assertions 4/8

- Enabling assertions:
  – Use the –enableassertions or –ea switch.
- **java –ea MyProgram**
  or
- **java –enableassertions MyProgram**

# Assertions 5/8

- To enable assertions add -ea as a VM option for your runtime configuration



# Assertions 6/8

```
1. public class TestAssert {
2.     public static void main(String[] arg){
3.         int x = 10;
4.         int y = 10;
5.         assert x==y;
6.     }
7. }
```

Ok!

# Assertions 7/8

```
1. public class TestAssert {
2.     public static void main(String[] arg){
3.         int x = 10;
4.         int y = 20;
5.         assert x==y;
6.     }
7. }
```

```
Exception in thread "main" java.lang.AssertionError
        at
com.brainacad.oop1.testexcp.TestAssert.main(TestAs
sert.java:10
```

# Assertions 8/8

```
1. public class TestAssert {
2.     public static void main(String[] arg){
3.         int x = 10;
4.         int y = 20;
5.         assert x==y:"x and y must be equal!";;
6.     }
7. }
```

```
Exception in thread "main" java.lang.AssertionError:
x and y must be equal!
        at
com.brainacad.oop1.testexcp.TestAssert.main(TestAs
sert.java:10)
```