CSE 1325

Week of 10/24/2022

Instructor : Donna French

OOP Vocabulary

Polymorphism

describes the ability of different objects to be accessed by a common interface

Inheritance

describes the ability of objects to derive behavior patterns from other objects

These objects can then customize and add new unique characteristics

Encapsulation

describes the ability of objects to maintain their internal workings independently from the program they are used in

Classes **encapsulate** attributes and methods into objects created from those classes

Polymorphism

Once a class implements an interface, all objects of that class have an ***is-a*** relationship with the interface type, and all objects of the class are guaranteed to provide the functionality described by the interface.

This is true of all subclasses of that class as well.

Interfaces are particularly useful for assigning common functionality to possibly unrelated classes.

Allows objects of unrelated classes to be processed polymorphically—objects of classes that implement the same interface can respond to all of the interface method calls.

Polymorphism Example

Space Objects in a Video Game

A video game manipulates objects of classes `Martian`, `Venusian`, `Plutonian`, `SpaceShip` and `LaserBeam`. Each inherits from `SpaceObject` and overrides its `draw()` method.

A screen manager maintains a collection of references to objects of the various classes and periodically sends each object the same message — `draw()`.

Each object responds in a unique way.

Polymorphism Example

A `Martian` object might draw itself in red with green eyes and the appropriate number of antennae.

A `SpaceShip` object might draw itself as a bright silver flying saucer.

A `LaserBeam` object might draw itself as a bright red beam across the screen.

The same message (`draw()`) sent to a variety of objects has “many forms” of results.

Polymorphism Example

A screen manager might use polymorphism to facilitate adding new classes to a system with minimal modifications to the system’s code.

To add new objects to our video game:

Build a class that extends `SpaceObject` and provides its own draw method implementation.

When objects of that class appear in the `SpaceObject` collection, the screen-manager code invokes method `draw()`, exactly as it does for every other object in the collection, regardless of its type.

So the new objects simply “plug right in” without any modification of the screen manager code by the programmer.

Polymorphism

Polymorphism enables you to deal in generalities and let the execution-time environment handle the specifics.

You can tell objects to behave in manners appropriate to those object without knowing their specific types as long as they belong to the same inheritance hierarchy.

Polymorphism promotes extensibility.

Abstract Classes and Methods

Sometimes it’s useful to declare classes for which you never intend to create objects.

Used only as superclasses in inheritance hierarchies, so they are sometimes called abstract superclasses.

Cannot be used to instantiate objects—abstract classes are incomplete.

Abstract Classes and Methods

Subclasses must declare the “missing pieces” to become “concrete” classes, from which you can instantiate objects; otherwise, these subclasses, too, will be abstract.

If a subclass does not DO something to become concrete, then the subclass will be abstract.

An abstract class provides a superclass from which other classes can inherit and thus share a common design.

Abstract Classes and Methods

Classes that can be used to instantiate objects are called **concrete** classes.

Such classes provide implementations of every method they declare (some of the implementations can be inherited).

Abstract superclasses are too general to create real objects—they specify only what is common among subclasses.

Concrete classes provide the specifics that make it reasonable to instantiate objects.

Not all hierarchies contain abstract classes.

Abstract Classes and Methods

Programmers often write client code that uses only abstract superclass types to reduce client code’s dependencies on a range of subclass types.

You can write a method with a parameter of an abstract superclass type.

When called, such a method can receive an object of any concrete class that directly or indirectly extends the superclass specified as the parameter’s type.

Abstract classes sometimes constitute several levels of a hierarchy.

Abstract Classes and Methods

Do we really want to instantiate an object from `Shape`?

We did create `Poly`

```
Shape A = new Shape("Poly");
```

But then `Poly` did silly things like print an area of 0.

Abstract Classes and Methods

We don't need to instantiate objects from `Shape` – we just need `Shape` to be the superclass from which subclasses can inherit.

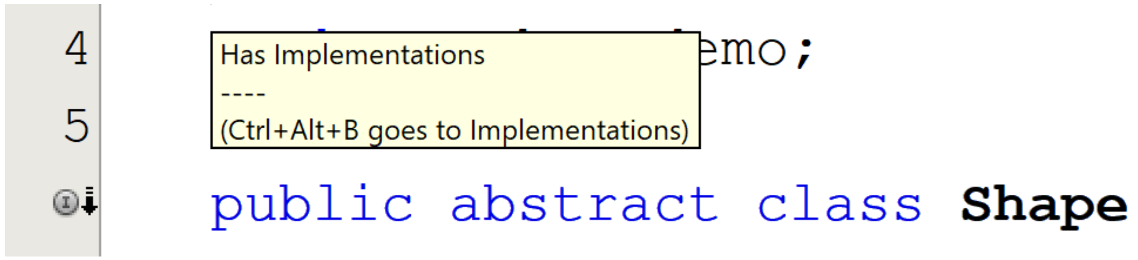
Sorry `Poly` – nice knowing you.

So how do we make a superclass `abstract?`

Abstract Classes and Methods

You make a class abstract by declaring it with keyword **abstract**.

```
public class Shape     public abstract class Shape
{
    private String shapeName;
    private double dim1;
    private double dim2;
    private String color;
```



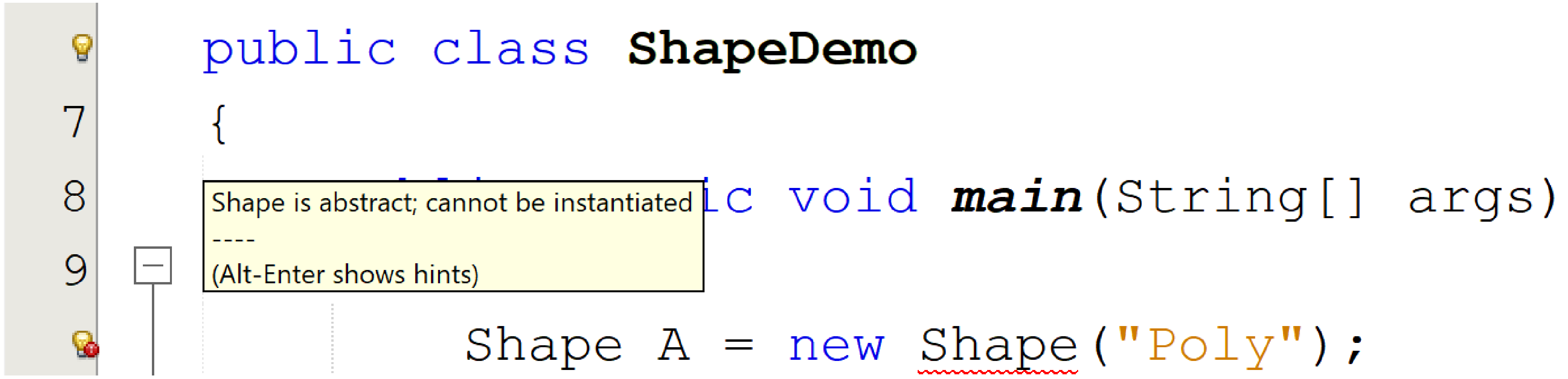
```
emo;
```

Has Implementations

(Ctrl+Alt+B goes to Implementations)

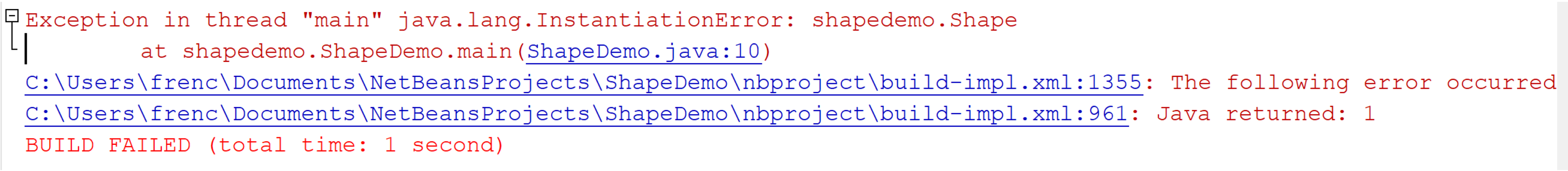
```
public abstract class Shape
```

Abstract Classes and Methods



```
public class ShapeDemo
{
                 ic void main(String[] args)

        Shape A = new Shape("Poly");
```

```
Shape is abstract; cannot be instantiated
----
(Alt-Enter shows hints)
```



```
Exception in thread "main" java.lang.InstantiationError: shapedemo.Shape
        at shapedemo.ShapeDemo.main(ShapeDemo.java:10)
C:\Users\frenc\Documents\NetBeansProjects\ShapeDemo\nbproject\build-impl.xml:1355: The following error occurred
C:\Users\frenc\Documents\NetBeansProjects\ShapeDemo\nbproject\build-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 1 second)
```

Abstract Classes and Methods

You make a class abstract by declaring it with keyword **`abstract`**.

An **`abstract`** class normally contains one or more **`abstract`** methods.

An **`abstract`** method is an instance method with keyword **`abstract`** in its declaration, as in

```
public abstract void draw(); // abstract method
```

Abstract methods do not provide implementations.

Abstract Classes and Methods

Remember the problem we had with `getArea()`?

When we moved it to `Shape()` to allow for polymorphism, we really did not know what to put in `getArea()`'s body in `Shape`

```
public double getArea()
{
    return 0;
}
```

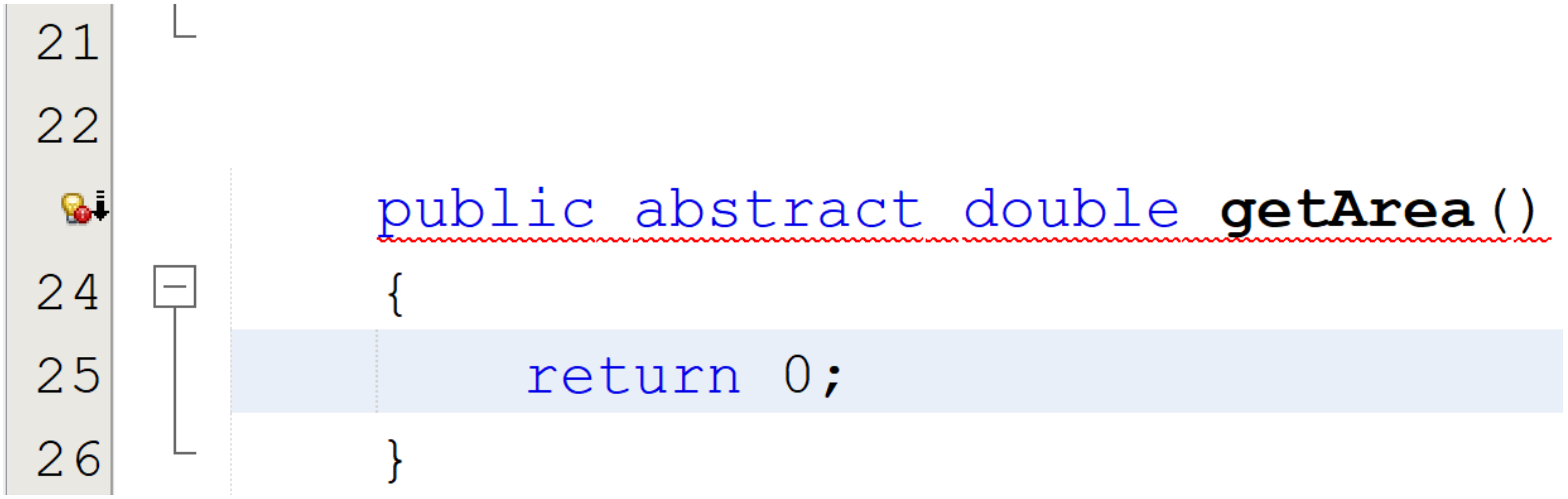
We added `return 0;` just because we HAD to have a return value because the method has a return value of `double`.

Abstract Classes and Methods

An **abstract** method is an instance method with keyword **abstract** in its declaration, as in

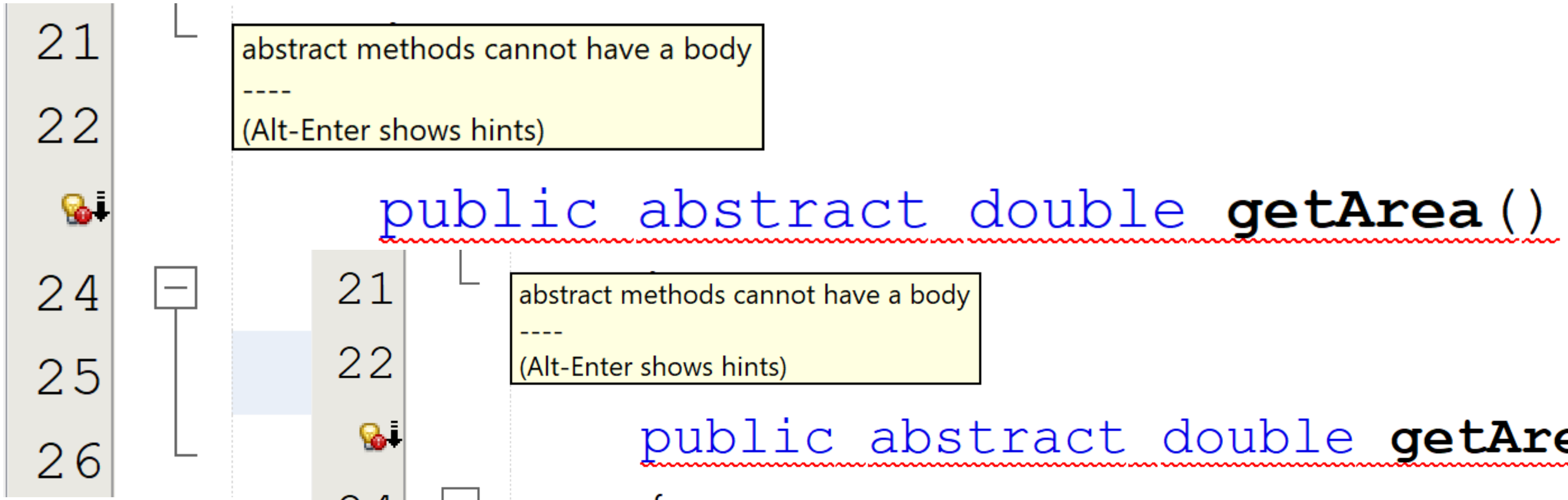
```
public abstract void draw(); // abstract method
```

Abstract methods do not provide implementations.



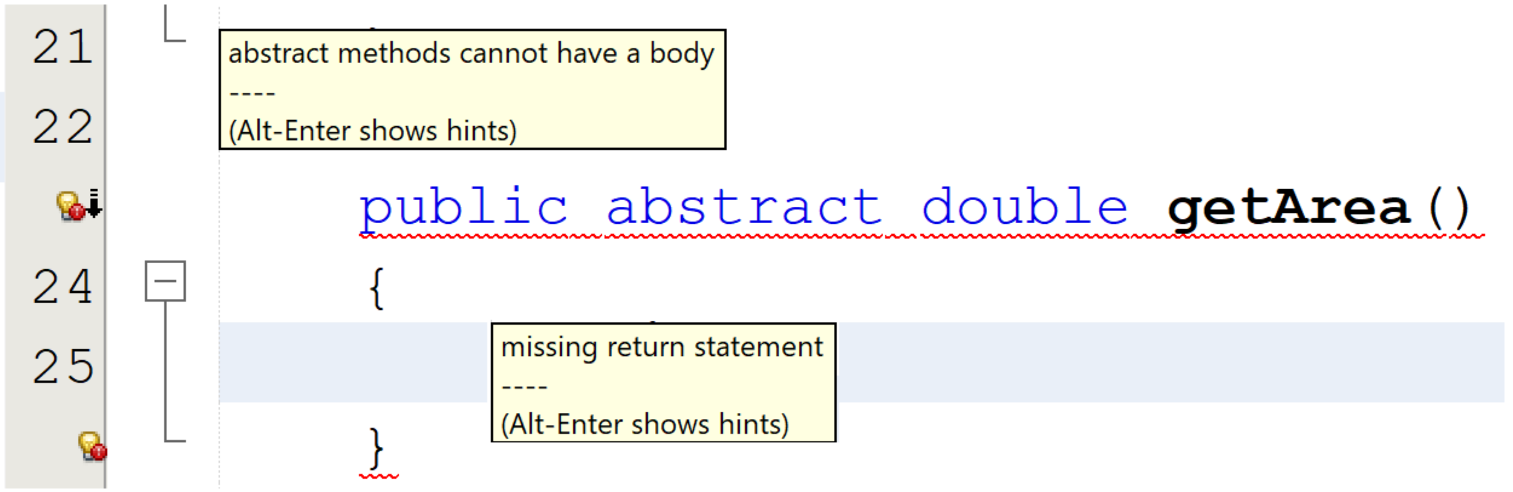
```
public abstract double getArea()
{
    return 0;
}
```

Abstract Classes and Methods



```
abstract methods cannot have a body
----
(Alt-Enter shows hints)
```

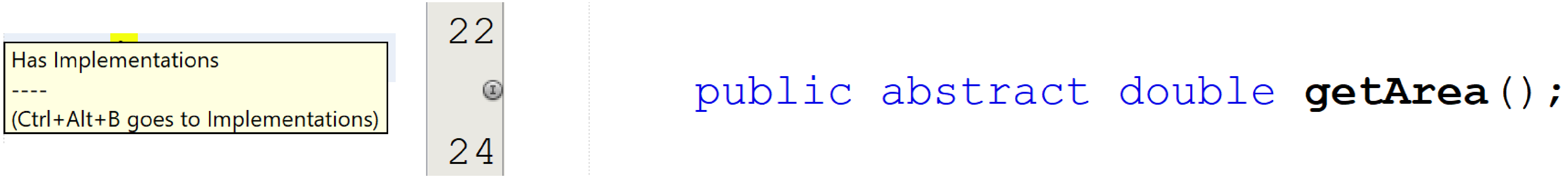
```java
    public abstract double getArea()
```



```
abstract methods cannot have a body
----
(Alt-Enter shows hints)
```

```java
    public abstract double getArea()
    {
                                        
    }
```

```
missing return statement
----
(Alt-Enter shows hints)
```

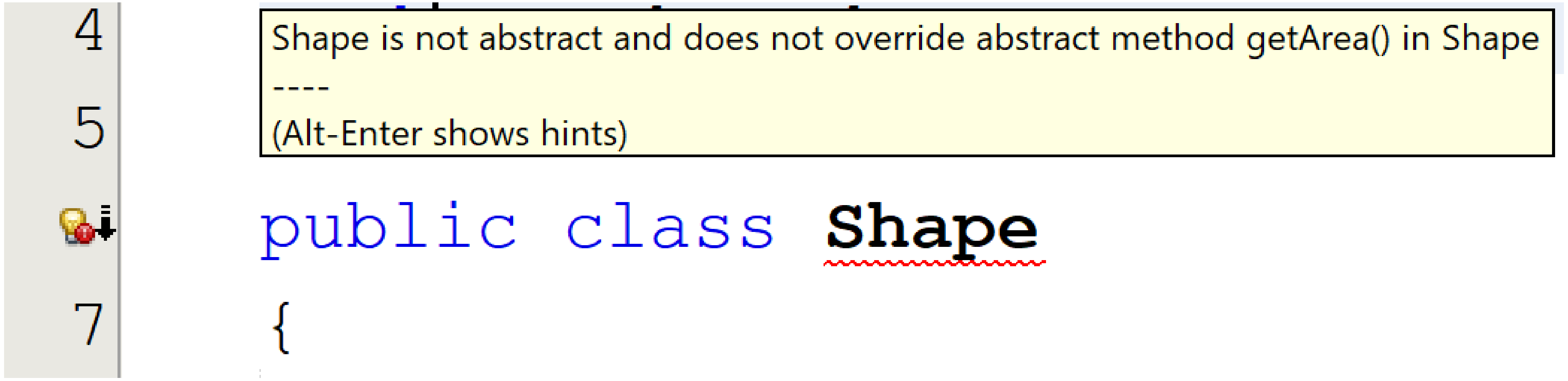


```
Has Implementations
----
(Ctrl+Alt+B goes to Implementations)
```

```java
    public abstract double getArea();
```

Abstract Methods and Classes

A class that contains abstract methods must be an abstract class even if that class contains some concrete (nonabstract) methods.



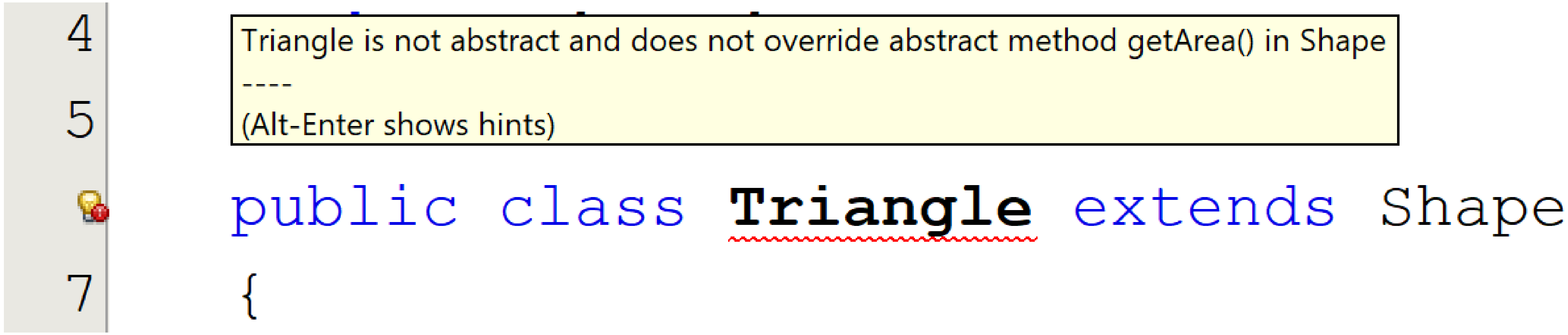
```
Shape is not abstract and does not override abstract method getArea() in Shape
----
(Alt-Enter shows hints)
```

```
public class Shape
{
```

Abstract Methods and Classes

Each concrete subclass of an abstract superclass also must provide concrete implementations of each of the superclass's abstract methods.

If we comment out `getArea()` in `Triangle`...



Triangle is not abstract and does not override abstract method getArea() in Shape
----
(Alt-Enter shows hints)

```
public class Triangle extends Shape
{
```

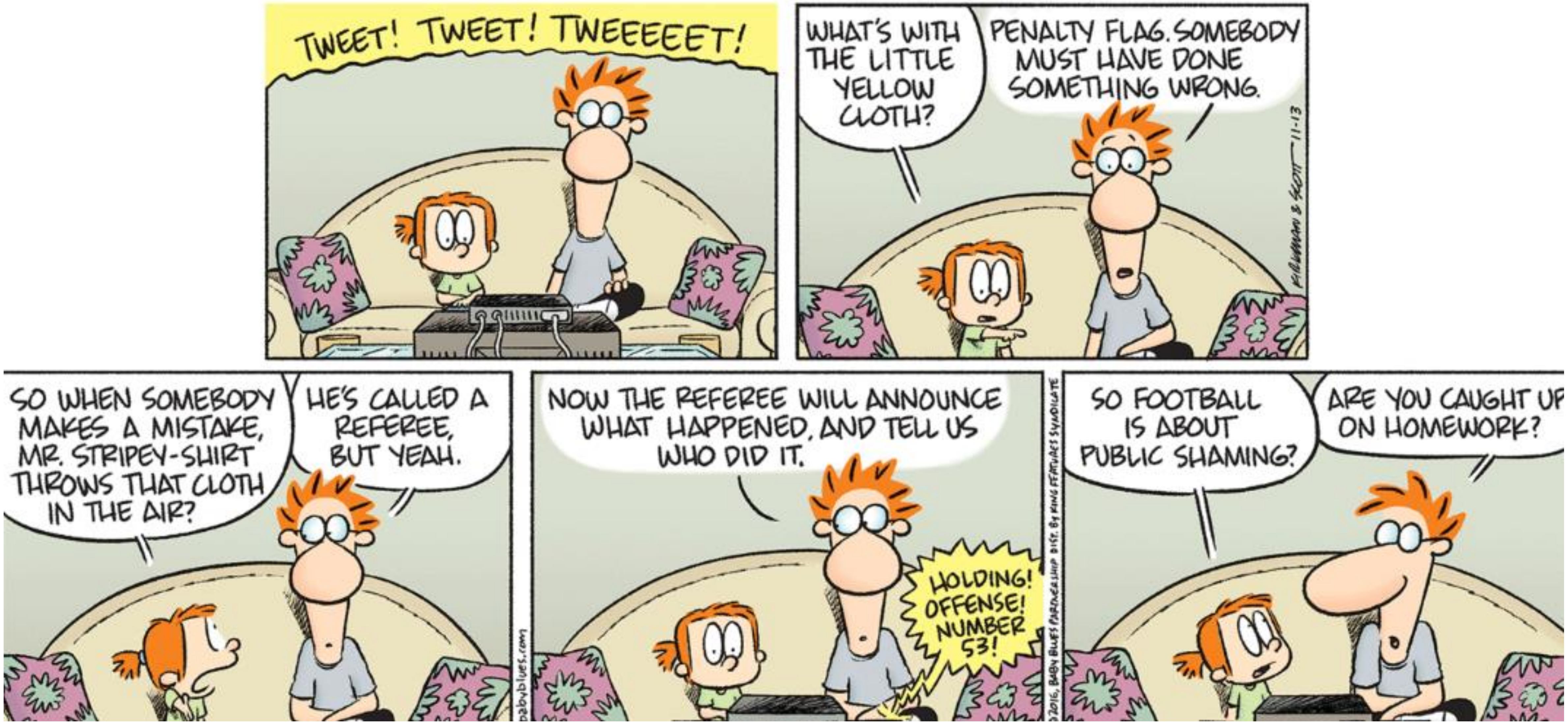
Abstract Methods and Classes

Constructors and static methods cannot be declared abstract.

An abstract class declares common attributes and behaviors (both abstract and concrete) of the classes in a class hierarchy.

An abstract class typically contains one or more abstract methods that subclasses must override if they want to be concrete.

The instance variables and concrete methods of an abstract class are subject to the normal rules of inheritance.


TWEET! TWEET! TWEEEEET!
WHAT'S WITH THE LITTLE YELLOW CLOTH?
PENALTY FLAG. SOMEBODY MUST HAVE DONE SOMETHING WRONG.
KIRKMAN & SCOTT 11-13
SO WHEN SOMEBODY MAKES A MISTAKE, MR. STRIPEY-SHIRT THROWS THAT CLOTH IN THE AIR?
HE'S CALLED A REFEREE, BUT YEAH.
babyblues.com
NOW THE REFEREE WILL ANNOUNCE WHAT HAPPENED, AND TELL US WHO DID IT.
HOLDING! OFFENSE! NUMBER 53!
©2016, BABY BLUES PARTNERSHIP DIST. BY KING FEATURES SYNDICATE
SO FOOTBALL IS ABOUT PUBLIC SHAMING?
ARE YOU CAUGHT UP ON HOMEWORK?

Exception Handling

When writing reusable code, error handling is a necessity. One of the most common ways to handle potential errors is via return codes.

The primary issue with return codes is that the error handling code ends up intricately linked to the normal control flow of the code. This in turn ends up constraining both how the code is laid out and how errors can be reasonably handled.

Exception handling provides a mechanism to decouple handling of errors or other exceptional circumstances from the typical control flow of your code. This allows more freedom to handle errors when and how ever is most useful for a given situation, alleviating many (if not all) of the messiness that return codes cause.

Introduction to Exception Handling

An **exception** indicates a problem that occurs while a program executes.

Should be a problem that occurs infrequently; hence; exception.

Exception handling allows you to create fault-tolerant programs that can handle exceptions.

- This may mean allowing the program to finish normally even if an exception occurred – like trying to access an out-of-range subscript in an array.

- More severe problems might require that the program notify the user of the problem and then terminate immediately.

```
public static int quotient(int numerator, int denominator)
{
    return numerator/denominator;
}

public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);

    System.out.print("Please enter an integer numerator : ");
    int numerator = in.nextInt();

    System.out.print("Please enter an integer denominator : ");
    int denominator = in.nextInt();

    int result = quotient(numerator, denominator);

    System.out.printf("\nResult : %d/%d = %d\n",
                        numerator, denominator, result);
}
```

```
Please enter an integer numerator : 100
Please enter an integer denominator : 20

Result : 100/20 = 5


Please enter an integer numerator : 100
Please enter an integer denominator : 0
Exception in thread "main" java.lang.ArithmeticException: / by
zero at
dividebyzerodemo.DivideByZeroDemo.quotient(DivideByZeroDemo.java:
12) at
dividebyzerodemo.DivideByZeroDemo.main(DivideByZeroDemo.java:25)
C:\Users\frenc\Documents\NetBeansProjects\DivideByZeroDemo\nbproj
ect\build-impl.xml:1355: The following error occurred while
executing this line:
C:\Users\frenc\Documents\NetBeansProjects\DivideByZeroDemo\nbproj
ect\build-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 9 seconds)
```

```
Please enter an integer numerator : 100
Please enter an integer denominator : hello
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at 
dividebyzerodemo.DivideByZeroDemo.main(DivideByZeroDemo.java:23
)
C:\Users\frenc\Documents\NetBeansProjects\DivideByZeroDemo\nbpr
oject\build-impl.xml:1355: The following error occurred while 
executing this line:
C:\Users\frenc\Documents\NetBeansProjects\DivideByZeroDemo\nbpr
oject\build-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 4 seconds)
```

Exception Handling

```
Please enter an integer numerator : 100
Please enter an integer denominator : 0

Exception in thread "main" java.lang.ArithmeticException


Please enter an integer numerator : 100
Please enter an integer denominator : hello

Exception in thread "main" java.util.InputMismatchException
```

```
try
{
    result = quotient(numerator, denominator);
}
catch (Exception e)
{
    System.out.println("Exception : Cannot divide by zero");
}
```

Moved declaration outside of `try`

```
Please enter an integer numerator : 100
Please enter an integer denominator : 0
Exception : Cannot divide by zero

Result : 100/0 = 0
```

```
public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);
    int result = 0;
    boolean hasException = false;
    int numerator = 0, denominator = 0;
    do
    {
        hasException = false;
        System.out.print("Please enter an integer numerator : ");
        numerator = in.nextInt();

        System.out.print("Please enter an integer denominator : ");
        denominator = in.nextInt();
        try
        {
            result = quotient(numerator, denominator);
        }
        catch (Exception e)
        {
            System.out.println("Exception : Cannot divide by zero");
            System.out.println("Reenter values");
            hasException = true;
        }

        if (!hasException)
            System.out.printf("\nResult : %d/%d = %d\n",
                                numerator, denominator, result);
    }
    while (hasException);
}
```

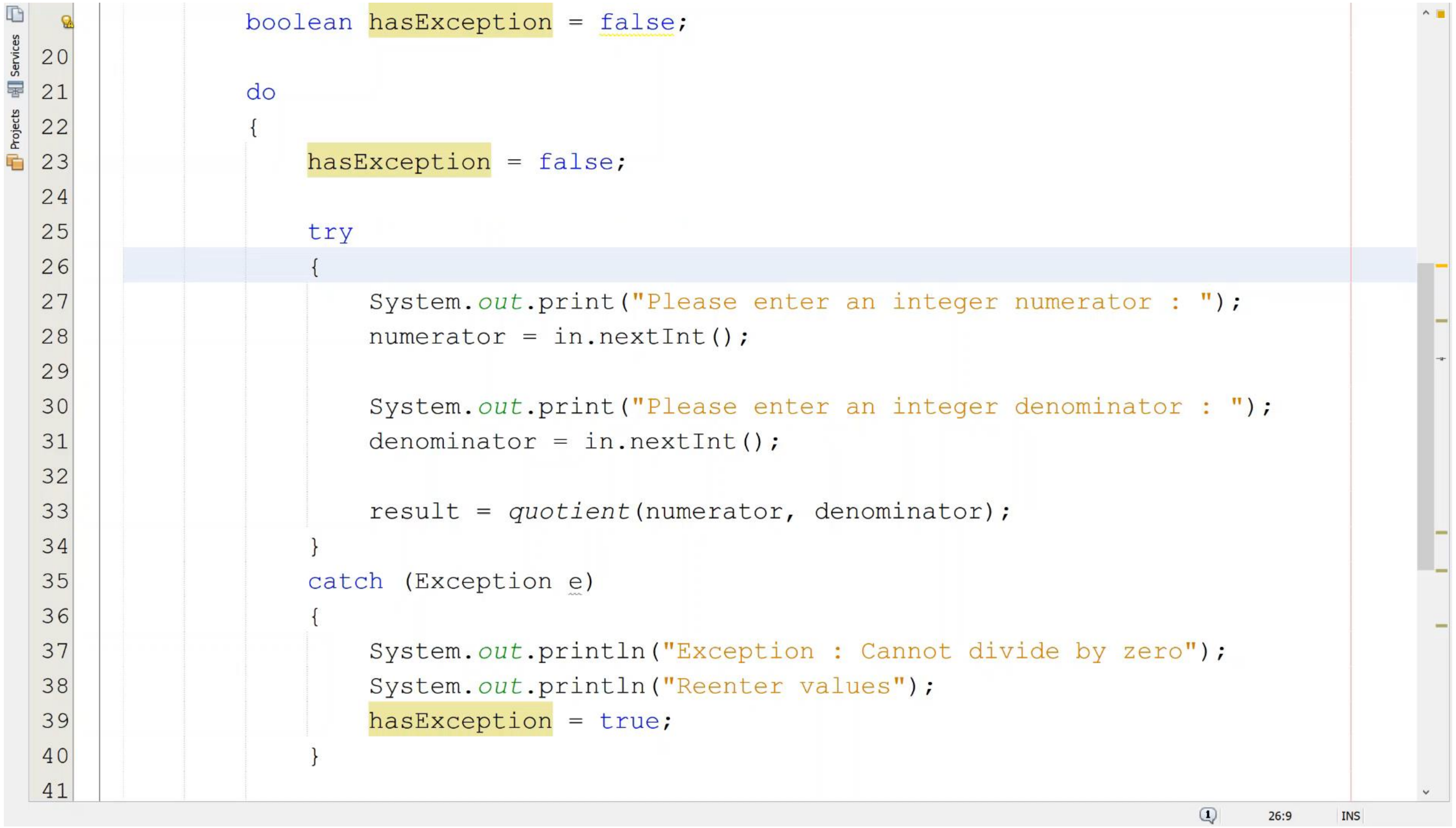
```
    do
    {
        hasException = false;
        System.out.print("Please enter an integer numerator : ");
        numerator = in.nextInt();

        System.out.print("Please enter an integer denominator : ");
        denominator = in.nextInt();

        try
        {
            result = quotient(numerator, denominator);
        }
        catch (Exception e)
        {
            System.out.println("Exception : Cannot divide by zero");
            System.out.println("Reenter values");
            hasException = true;
        }

        if (!hasException)
            System.out.printf("\nResult : %d/%d = %d\n", numerator, denominator, result);
    }
    while (hasException);
}
```

```
Please enter an integer numerator : 100
Please enter an integer denominator : 10

Result : 100/10 = 10
```

```
Please enter an integer numerator : 100
Please enter an integer denominator : 0
Exception : Cannot divide by zero
Reenter values
Please enter an integer numerator : 100
Please enter an integer denominator : 1

Result : 100/1 = 100
```

```
Please enter an integer numerator : 100
Please enter an integer denominator : hello
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at
dividebyzerodemo.DivideByZeroDemo.main(DivideByZeroDemo.java:28)
C:\Users\frenc\Documents\NetBeansProjects\DivideByZeroDemo\nbproject
\build-impl.xml:1355: The following error occurred while executing
this line:
C:\Users\frenc\Documents\NetBeansProjects\DivideByZeroDemo\nbproject
\build-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 4 seconds)
```

```
do
{
    hasException = false;
    try
    {
        System.out.print("Please enter an integer numerator : ");
        numerator = in.nextInt();

        System.out.print("Please enter an integer denominator : ");
        denominator = in.nextInt();

        result = quotient(numerator, denominator);
    }
```



```
        boolean hasException = false;

        do
        {
            hasException = false;

            try
            {
                System.out.print("Please enter an integer numerator : ");
                numerator = in.nextInt();

                System.out.print("Please enter an integer denominator : ");
                denominator = in.nextInt();

                result = quotient(numerator, denominator);
            }
            catch (Exception e)
            {
                System.out.println("Exception : Cannot divide by zero");
                System.out.println("Reenter values");
                hasException = true;
            }

```

```
catch (Exception e)
{
    System.out.println("Exception : Cannot divide by zero");
    System.out.println("Reenter values");
    in.nextLine();
    hasException = true;
}
```

```
Please enter an integer numerator : 100
Please enter an integer denominator : 0
Exception : Cannot divide by zero
Reenter values
Please enter an integer numerator : 100
Please enter an integer denominator : zero
Exception : Cannot divide by zero
Reenter values
Please enter an integer numerator : hello
Exception : Cannot divide by zero
Reenter values
```

```
Please enter an integer
numerator : 100
Please enter an integer
denominator : 9

Result : 100/9 = 11
```

```
Please enter an integer numerator : 100
Please enter an integer denominator : 0
Exception : Cannot divide by zero
Reenter values
Please enter an integer numerator : 100
Please enter an integer denominator : zero
Exception : Cannot divide by zero
```

Should entering "`zero`" or any other non integer value trigger a message of

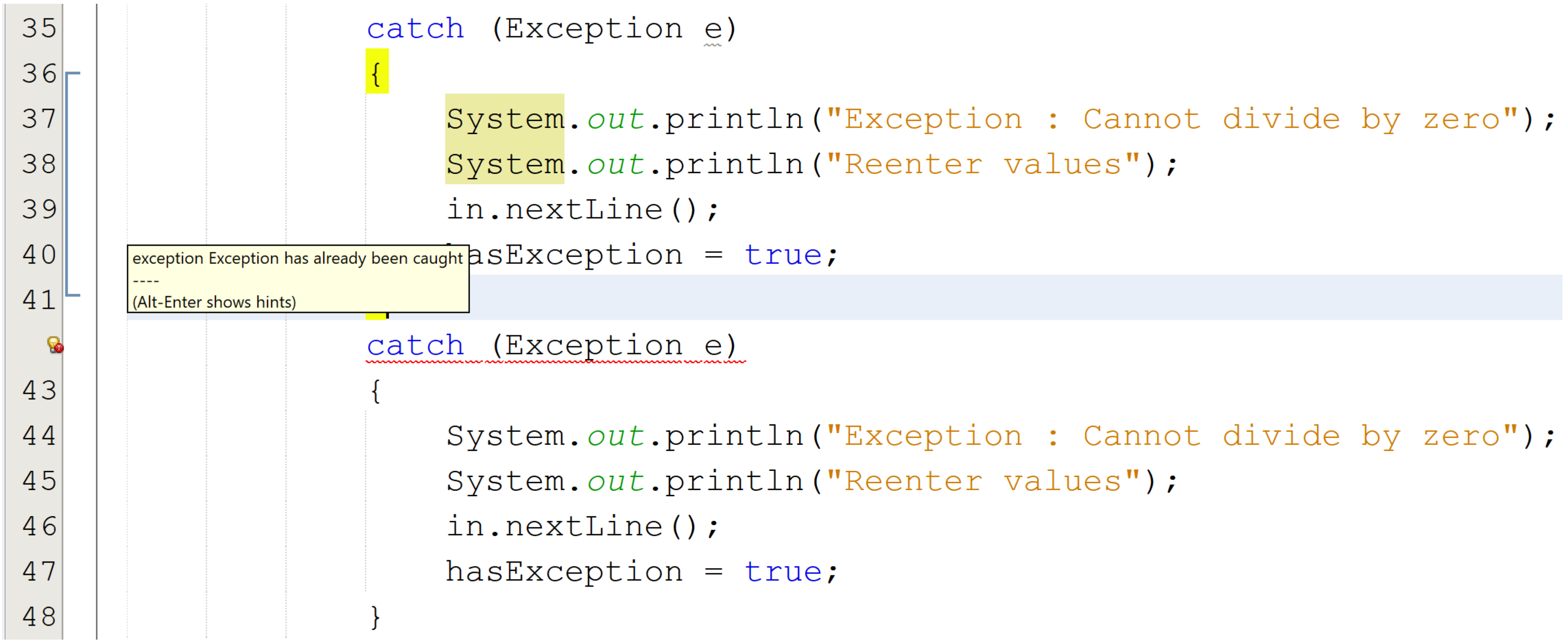
```
Exception : Cannot divide by zero
```

??

Exception Handling

```
Please enter an integer numerator : 100
Please enter an integer denominator : 0

Exception in thread "main" java.lang.ArithmeticException


Please enter an integer numerator : 100
Please enter an integer denominator : hello

Exception in thread "main" java.util.InputMismatchException
```

```
catch (ArithmeticException e)
{
    System.out.println("Exception : Cannot divide by zero");
    System.out.println("Reenter values");
    hasException = true;
}
catch (InputMismatchException e)
{
    System.out.println("Exception : You must enter integers ");
    System.out.println("Reenter values");
    in.nextLine();
    hasException = true;
}
```



```
            catch (Exception e)
            {
                System.out.println("Exception : Cannot divide by zero");
                System.out.println("Reenter values");
                in.nextLine();
                hasException = true;
            }
            catch (Exception e)
            {
                System.out.println("Exception : Cannot divide by zero");
                System.out.println("Reenter values");
                in.nextLine();
                hasException = true;
            }
```

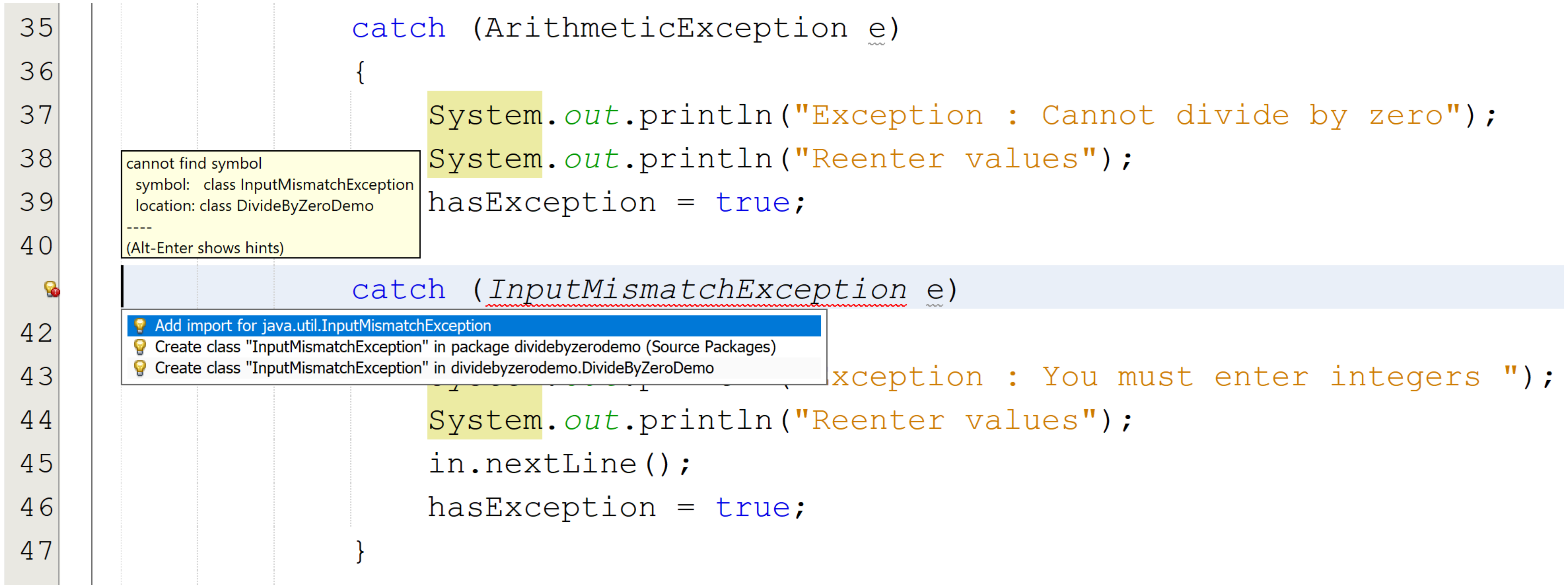
exception Exception has already been caught

(Alt-Enter shows hints)

```
Please enter an integer numerator : 100
Please enter an integer denominator   0
Exception : Cannot divide by zero
Reenter values


Please enter an integer numerator : 100
Please enter an integer denominator :   ro
Exception : You must enter integers


Reenter values
Please enter an integer numerator : 100
Please enter an integer denominator : 8

Result : 100/8 = 12
```

ArithmeticException

InputMismatchException

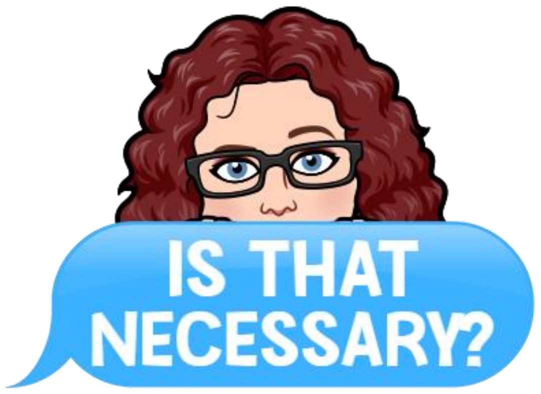


```
            catch (ArithmeticException e)
            {
                System.out.println("Exception : Cannot divide by zero");
                System.out.println("Reenter values");
                hasException = true;

            catch (InputMismatchException e)

                ...xception : You must enter integers ");
                System.out.println("Reenter values");
                in.nextLine();
                hasException = true;
            }
```

```
cannot find symbol
  symbol:   class InputMismatchException
  location: class DivideByZeroDemo
----
(Alt-Enter shows hints)
```

```
Add import for java.util.InputMismatchException
Create class "InputMismatchException" in package dividebyzerodemo (Source Packages)
Create class "InputMismatchException" in dividebyzerodemo.DivideByZeroDemo
```

Exception Handling



```
    catch (ArithmeticException e)
    {
        System.out.println("Exception : Cannot divide by zero");
        System.out.println("Reenter values");
        hasException = true;
    }

    if (!hasException)
        System.out.printf("\nResult : %d/%d = %d\n",
                                    numerator, denominator, result);
}
while (hasException);
```

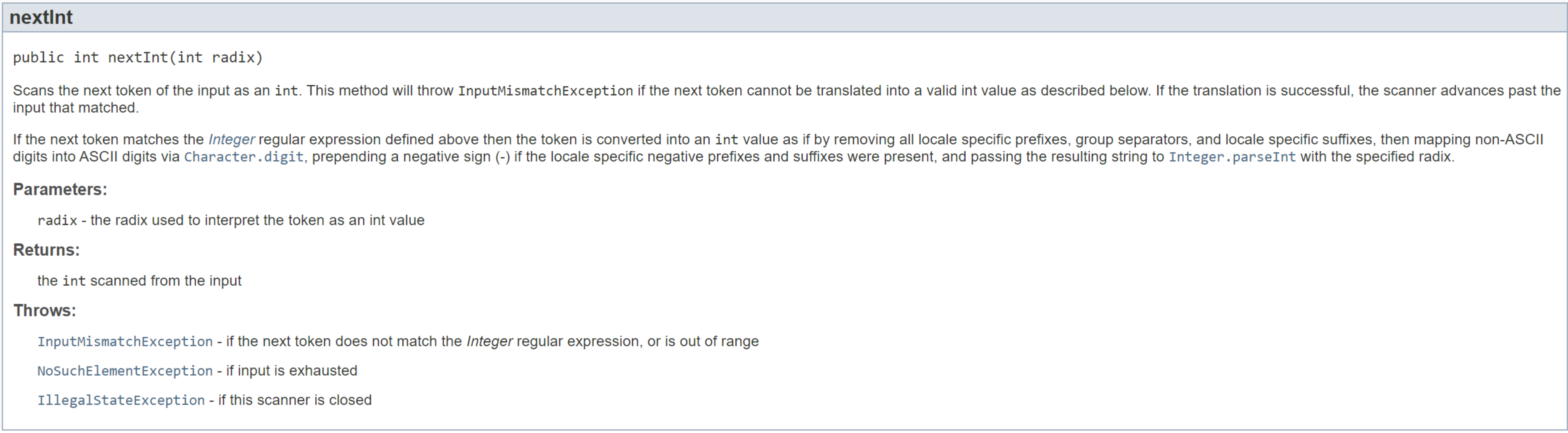
While we could detect if the `denominator` is 0, the code is actually longer and messier to get the same result.

Exception Handling

Read a method's documentation before using it to determine if the method can throw an exception and what exceptions are thrown.

Putting a method inside a try when the method cannot throw an exception is poor coding (indicates that you do not understand what your methods are doing).

Look at what exceptions can be thrown and determine if you need to add code to catch that particular exception.



nextInt

```
public int nextInt(int radix)
```

Scans the next token of the input as an `int`. This method will throw `InputMismatchException` if the next token cannot be translated into a valid int value as described below. If the translation is successful, the scanner advances past the input that matched.

If the next token matches the *Integer* regular expression defined above then the token is converted into an `int` value as if by removing all locale specific prefixes, group separators, and locale specific suffixes, then mapping non-ASCII digits into ASCII digits via `Character.digit`, prepending a negative sign (-) if the locale specific negative prefixes and suffixes were present, and passing the resulting string to `Integer.parseInt` with the specified radix.

Parameters:

`radix` - the radix used to interpret the token as an int value

Returns:

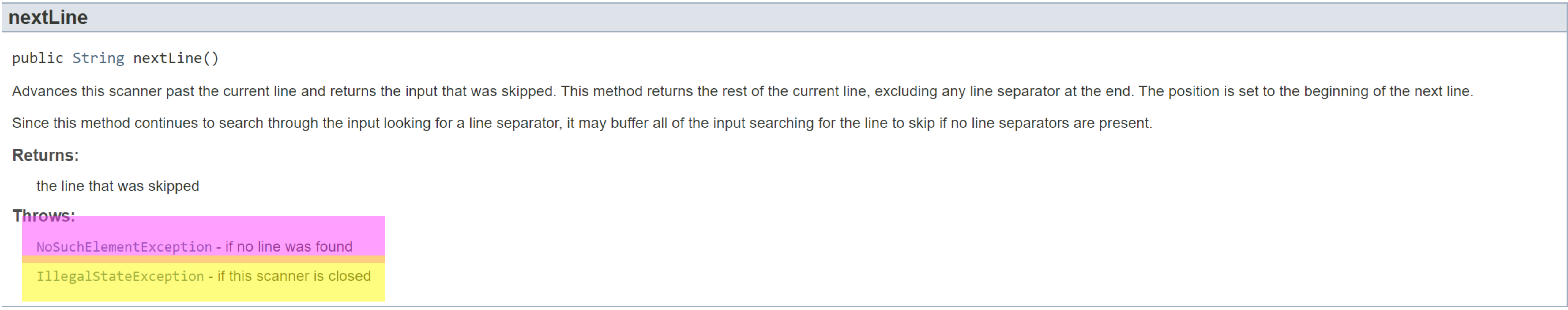
the `int` scanned from the input

Throws:

`InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range

`NoSuchElementException` - if input is exhausted

`IllegalStateException` - if this scanner is closed



nextLine

```
public String nextLine()
```

Advances this scanner past the current line and returns the input that was skipped. This method returns the rest of the current line, excluding any line separator at the end. The position is set to the beginning of the next line.

Since this method continues to search through the input looking for a line separator, it may buffer all of the input searching for the line to skip if no line separators are present.

Returns:

the line that was skipped

Throws:

`NoSuchElementException` - if no line was found

`IllegalStateException` - if this scanner is closed

`nextInt()` will throw an exception is a non integer value is entered – that is an easy mistake for a user to make.

`nextLine()` will take just about any input – numeric or alpha.

Notice when `nextLine()` will throw an exception…

Should we put a try-catch around every `nextLine()` just in case the `Scanner` is closed??

Not usually. We'll talk more about this later….

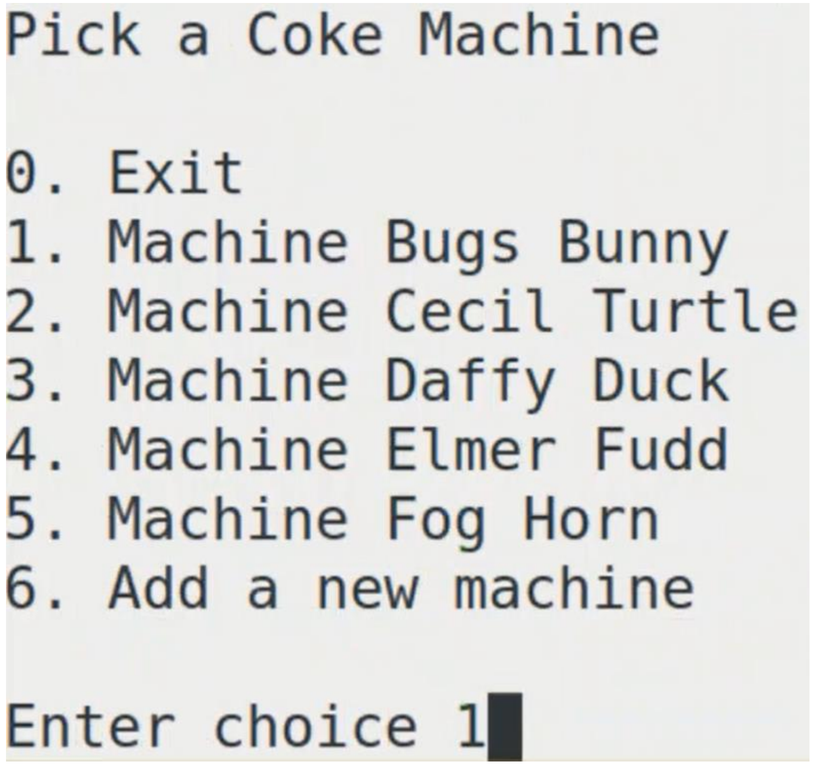
Coding Assignment 4

- Add command line parameters
  - read in a file
    - IFILENAME=xxxxxx
  - write out a file
    - OFILENAME=xxxxx
- Parse file of pipe delimited Coke Machines information using `split()`
  - name|price|change|inventory
- Create and manipulate an `ArrayList` of Coke Machines objects
- Display menu of Coke Machines and allow operations on each machine
- Exception handling
- Default constructor
- Overload `toString()` to print object

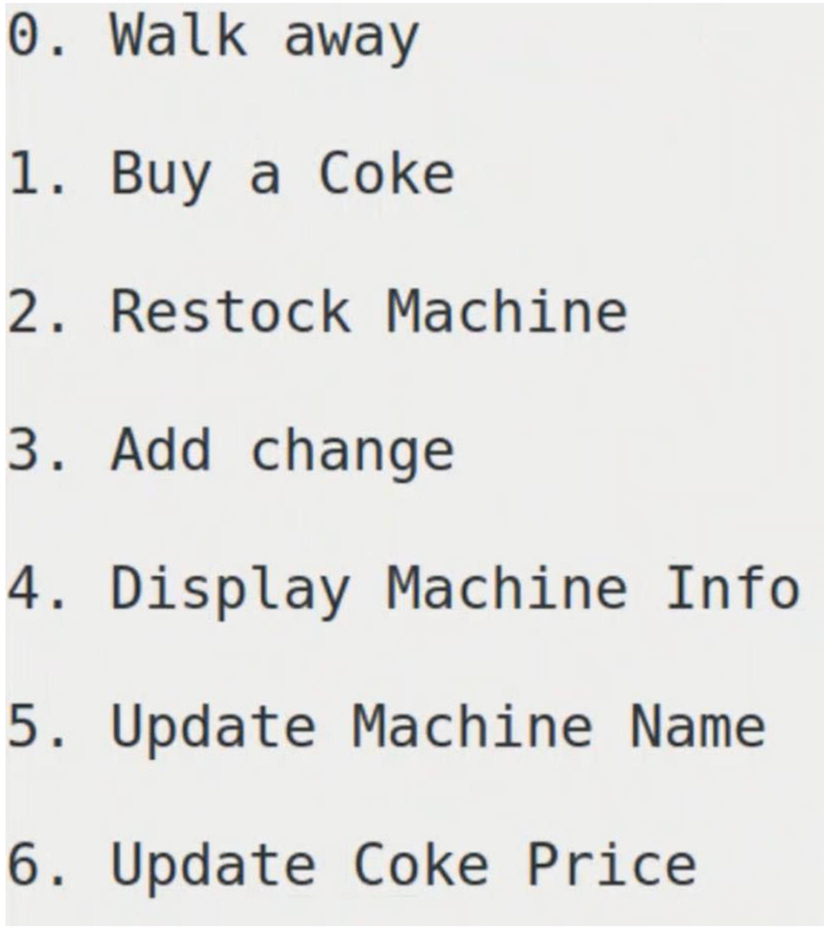
Coding Assignment 4

```
Machine Bugs Bunny|50|500|50
Machine Cecil Turtle|45|545|45
Machine Daffy Duck|40|540|1
Machine Elmer Fudd|100|1000|10
Machine Fog Horn|35|350|99
```

Coding Assignment 4



```
Pick a Coke Machine

0. Exit
1. Machine Bugs Bunny
2. Machine Cecil Turtle
3. Machine Daffy Duck
4. Machine Elmer Fudd
5. Machine Fog Horn
6. Add a new machine

Enter choice 1
```



```
0. Walk away

1. Buy a Coke

2. Restock Machine

3. Add change

4. Display Machine Info

5. Update Machine Name

6. Update Coke Price
```

Multi-Catch

It's relatively common for a try block to be followed by several catch blocks to handle various types of exceptions.

If the bodies of several catch blocks are identical, you can use the multi-catch feature to catch those exception types in a single catch

```
catch(Type1 | Type2 | Type3 exceptionobjectname)
```

```
public class ExceptionHandling
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int Cat = 0;
        int Dog[] = new int[10];

        System.out.print("Enter a number ");

        try
        {
            Cat = in.nextInt();
            Dog[11] = Cat;
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.printf("\nCaught it! %s\n", e);
        }
        catch(InputMismatchException e)
        {
            System.out.printf("\nCaught it! %s\n", e);
        }
    }
}
```

```
Enter a number a

Caught it!
java.util.InputMismatchException



Enter a number 1

Caught it!
java.lang.ArrayIndexOutOfBoundsExce
ption: Index 11 out of bounds for
length 10
```

```
public class ExceptionHandling
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int Cat = 0;
        System.out.print("Enter a number ");

        try
        {
            Cat = in.nextInt();
        }
        catch(ArithmeticException | InputMismatchException e)
        {
            System.out.printf("\nCaught it! %s\n", e);
        }
    }
}
```

```
Enter a number 1

Caught it! java.lang.ArrayIndexOutOfBoundsException: Index 11
out of bounds for length 10

Enter a number a

Caught it! java.util.InputMismatchException
```

```
catch(ArithmeticException | InputMismatchException e)
{
   System.out.printf("\nCaught it! %s\n", e);
}
```

```
try
{
    Cat = in.nextInt();
    Dog[9] = Cat;
    Cat = Cat/0;
}
catch(ArrayIndexOutOfBoundsException | InputMismatchException e)
{
    System.out.printf("\nCaught it! %s\n", e);
}
```

What is going to happen if I enter a number at the prompt?

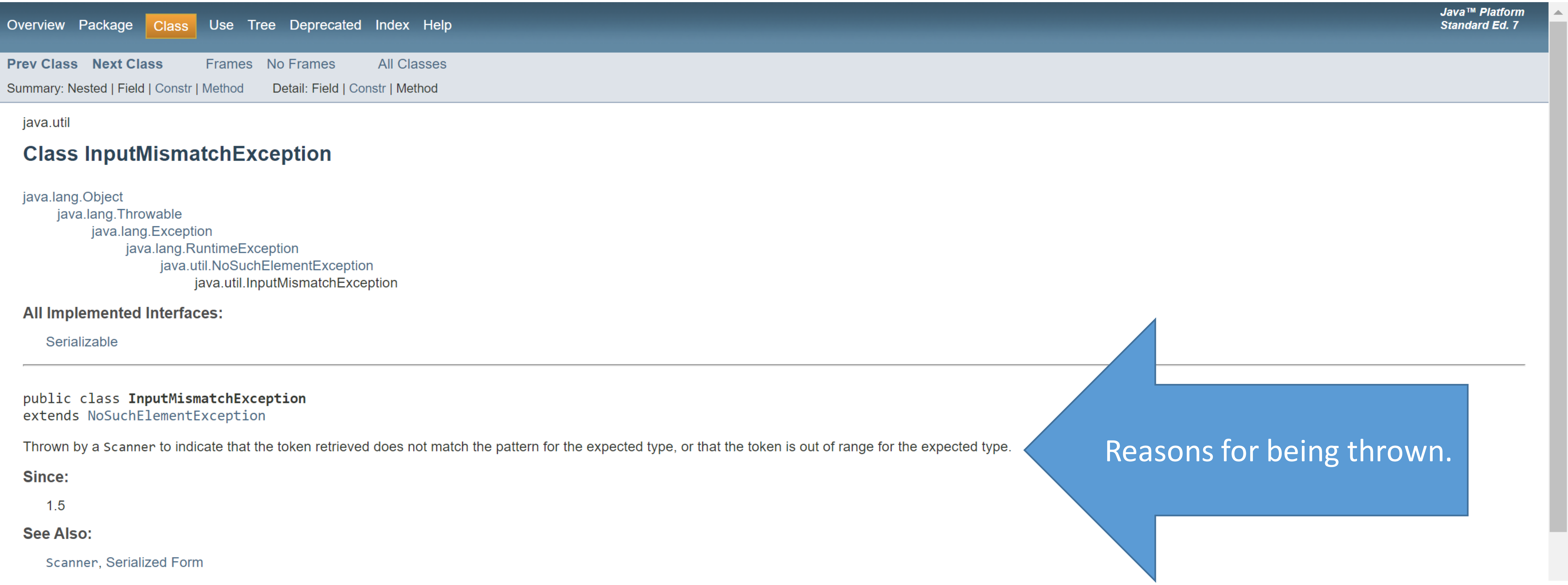
```
Enter a number 1
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at exceptionhandling.ExceptionHandling.main(ExceptionHandling.java:23)
C:\Users\frenc\Documents\NetBeansProjects\ExceptionHandling\nbproject\build-
impl.xml:1355: The following error occurred while executing this line:
C:\Users\frenc\Documents\NetBeansProjects\ExceptionHandling\nbproject\build-
impl.xml:961: Java returned: 1
```

Exception Handling

Read an exception class's documentation before using it to determine what type of exception that class can catch.

The documentation typically contains potential reasons that the exception would be thrown.

Generally, you should try to catch the most specific exception possible in order to obtain the most information you can about the exception.


Overview
Package
Class
Use
Tree
Deprecated
Index
Help
Java™ Platform Standard Ed. 7
Prev Class
Next Class
Frames
No Frames
All Classes
Summary: Nested | Field | Constr | Method
Detail: Field | Constr | Method

java.util

Class InputMismatchException

java.lang.Object
java.lang.Throwable
java.lang.Exception
java.lang.RuntimeException
java.util.NoSuchElementException
java.util.InputMismatchException

All Implemented Interfaces:

Serializable

```
public class InputMismatchException
extends NoSuchElementException
```

Thrown by a `Scanner` to indicate that the token retrieved does not match the pattern for the expected type, or that the token is out of range for the expected type.

Since:

1.5

See Also:

`Scanner`, Serialized Form

Constructor Summary

Constructors

Constructor and Description
`InputMismatchException()`   Constructs an `InputMismatchException` with `null` as its error message string.
`InputMismatchException(String s)`   Constructs an `InputMismatchException`, saving a reference to the error message string `s` for later retrieval by the `getMessage` method.

Method Summary

Methods inherited from class java.lang.Throwable

`addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

InputMismatchException

```
public InputMismatchException()
```

Constructs an `InputMismatchException` with `null` as its error message string.

InputMismatchException

```
public InputMismatchException(String s)
```

Constructs an `InputMismatchException`, saving a reference to the error message string `s` for later retrieval by the `getMessage` method.

Parameters:

`s` - the detail message.

```
package arrayoutofbounds;

public class ArrayOutOfBounds
{
    public static void PrintArray(int x[])
    {
        for (int i = 0; i <= x.length; i++)
        {
            System.out.print(x[i]);
        }
        System.out.println();
    }

    public static void main(String[] args)
    {
        int x[] = {0,1,2,3,4,5,6,7,8,9};

        PrintArray(x);
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Index 10 out of bounds for length 10
0123456789 at
arrayoutofbounds.ArrayOutOfBounds.PrintArray(ArrayOutOfBounds.java:12)
     at
arrayoutofbounds.ArrayOutOfBounds.main(ArrayOutOfBounds.java:21)
C:\Users\frenc\Documents\NetBeansProjects\ArrayOutOfBounds\nbproject\b
uild-impl.xml:1355: The following error occurred while executing this
line:
C:\Users\frenc\Documents\NetBeansProjects\ArrayOutOfBounds\nbproject\b
uild-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 4 seconds)
```

```
package arrayoutofbounds;

public class ArrayOutOfBounds
{
    public static void PrintArray(int x[])
    {
        for (int i = 0; i <= x.length; i++)
        {
            System.out.print(x[i]);
        }
        System.out.println();
    }

    public static void main(String[] args)
    {
        int x[] = {0,1,2,3,4,5,6,7,8,9};

        PrintArray(x);
    }
}
```

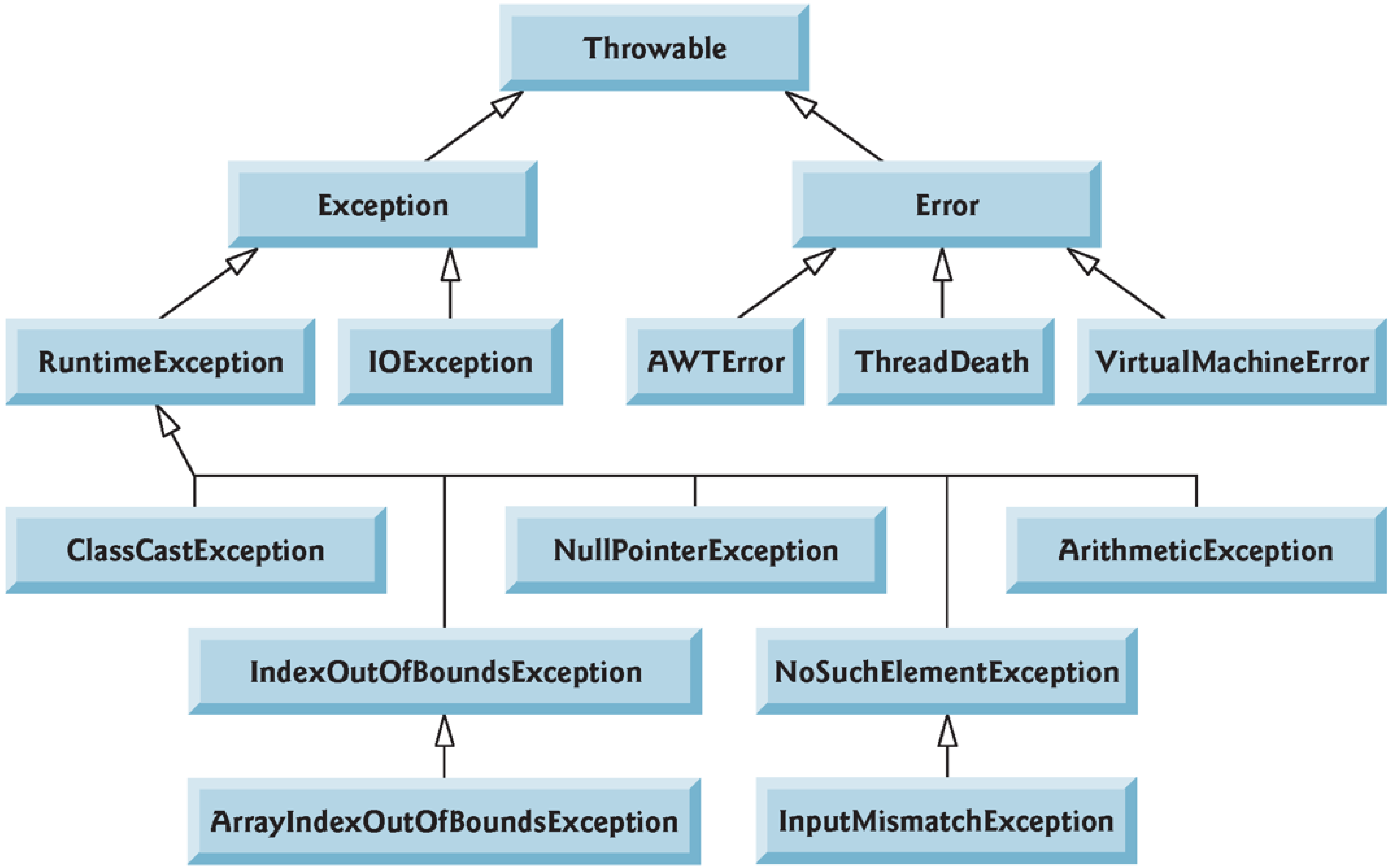
```
try
{
    PrintArray(x);
}
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println(e);
}

catch (IndexOutOfBoundsException e)

catch (RuntimeException e)

catch (Exception e)

catch (Throwable e)
```

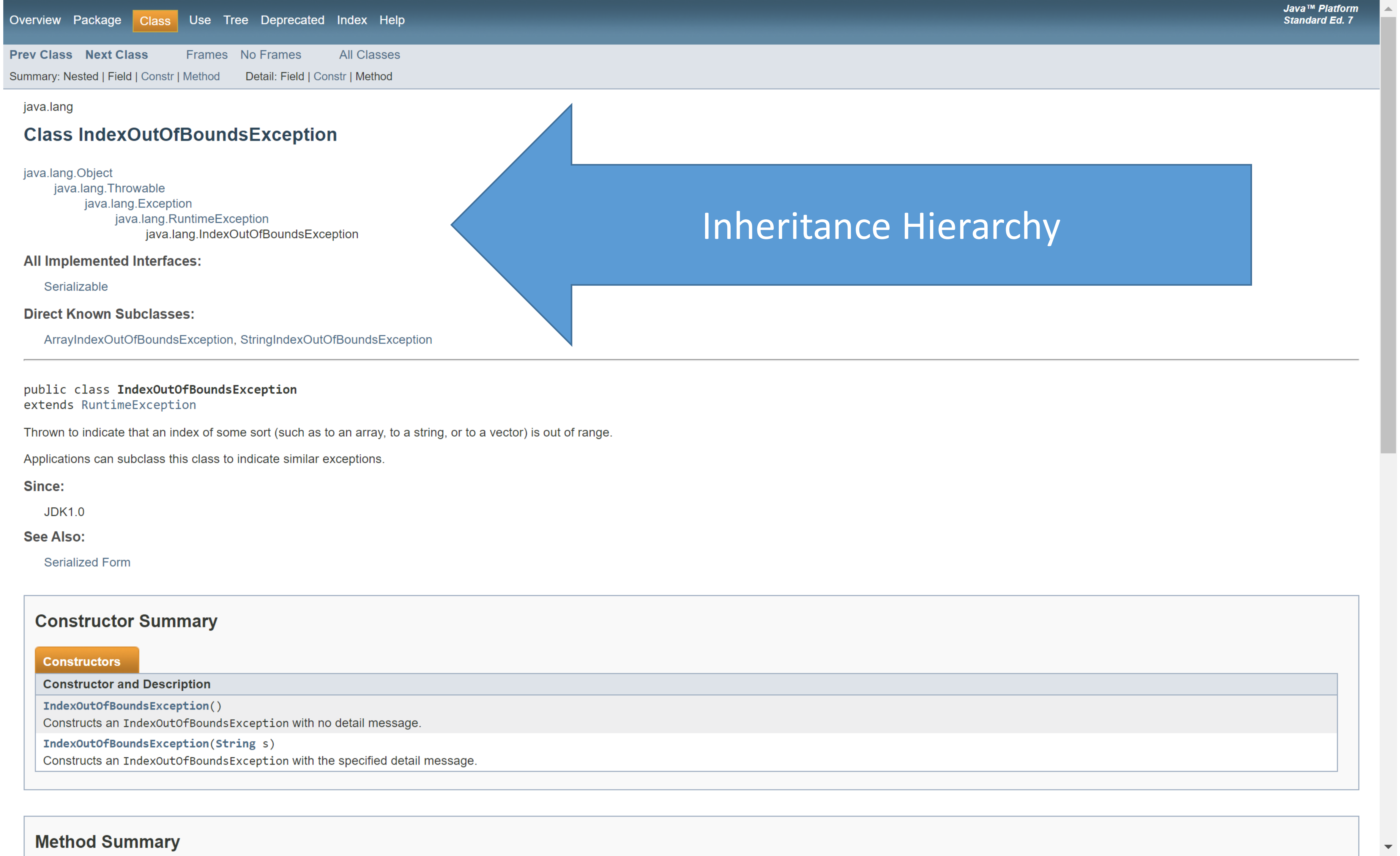
Most specific

Least specific

Any of these exceptions will catch that same exception.

Why?


Throwable
Exception
Error
RuntimeException
IOException
AWTError
ThreadDeath
VirtualMachineError
ClassCastException
NullPointerException
ArithmeticException
IndexOutOfBoundsException
NoSuchElementException
ArrayIndexOutOfBoundsException
InputMismatchException


Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class IndexOutOfBoundsException

java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.RuntimeException
                java.lang.IndexOutOfBoundsException

Inheritance Hierarchy

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException

```
public class IndexOutOfBoundsException
extends RuntimeException
```

Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

Applications can subclass this class to indicate similar exceptions.

Since:

JDK1.0

See Also:

Serialized Form

Constructor Summary

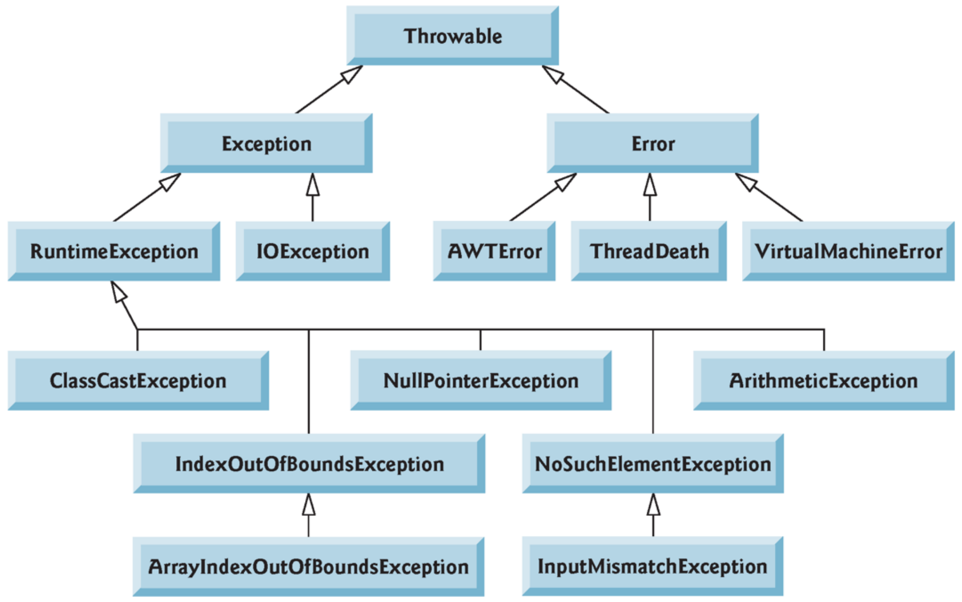
Constructors

Constructor and Description
`IndexOutOfBoundsException()` Constructs an `IndexOutOfBoundsException` with no detail message.
`IndexOutOfBoundsException(String s)` Constructs an `IndexOutOfBoundsException` with the specified detail message.

Method Summary

Exceptions

All Java exceptions classes inherit directly or indirectly from class `Exception`.



You can extend this hierarchy with your own exception clauses.

Exceptions

Only `Throwable` objects can be used with the exception-handling mechanism.

Class `Throwable` has two direct subclasses

`Exception`

represents exceptional situations that can occur in a Java program that can be caught by the JVM

`Error`

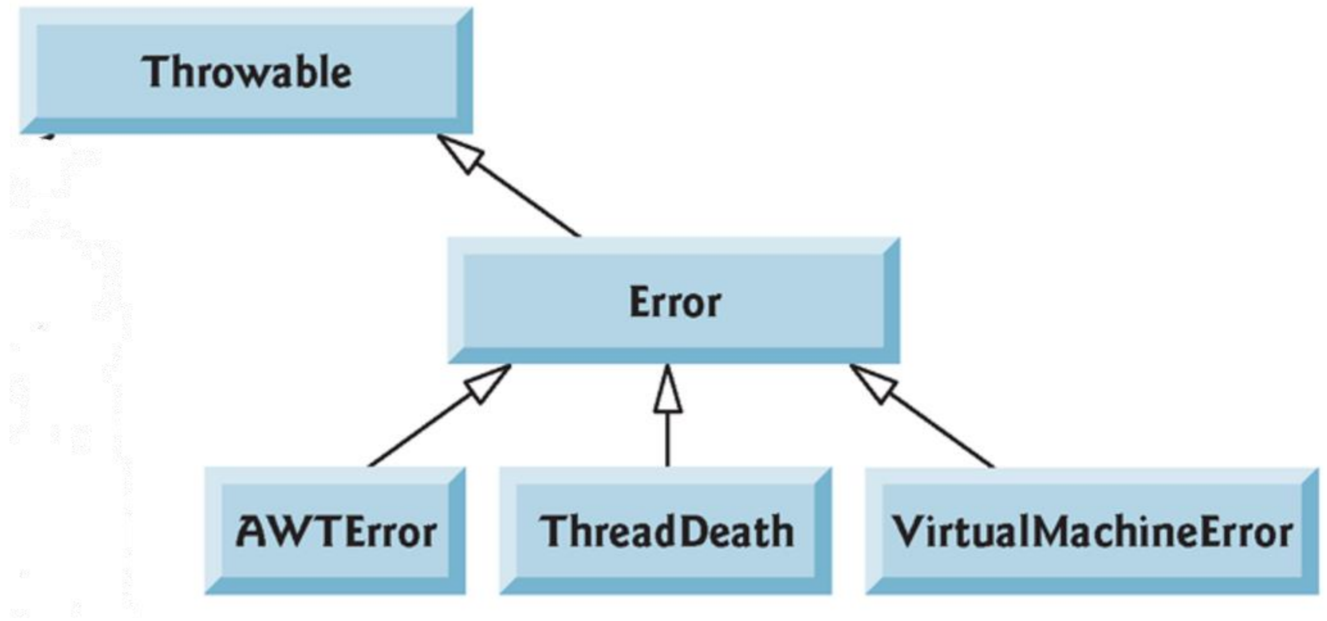
represents abnormal situations that happen in the JVM

Exceptions

`Error`

These exceptions should happen infrequently and should not be caught by applications

It's usually not possible for applications to recover from `Error`s.



Exception Handling

Checked vs Uncheck Exceptions

Java distinguishes between checked and unchecked exceptions.

The Java compiler enforces special requirements for checked exceptions.

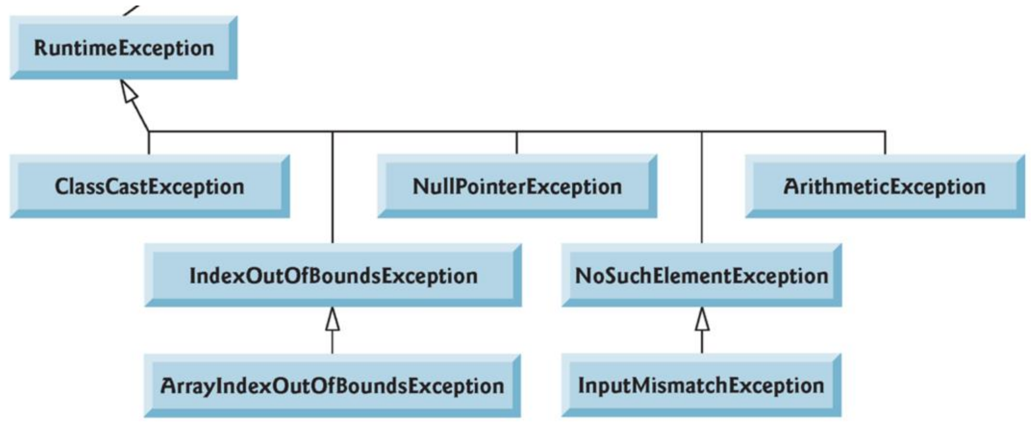
An exception's type determines whether it's checked or unchecked.

Exception Handling

All exceptions that are direct or indirect subclasses of

`RunTimeExceptions`

are unchecked exceptions.



Unchecked exceptions are typically exceptions caused by defects in your program's code.

```
ArrayIndexOutOfBoundsException
ArithmeticExceptions
```

```
public static int quotient(int numerator, int denominator) throws ArithmeticException
{
    return numerator/denominator;
}
```

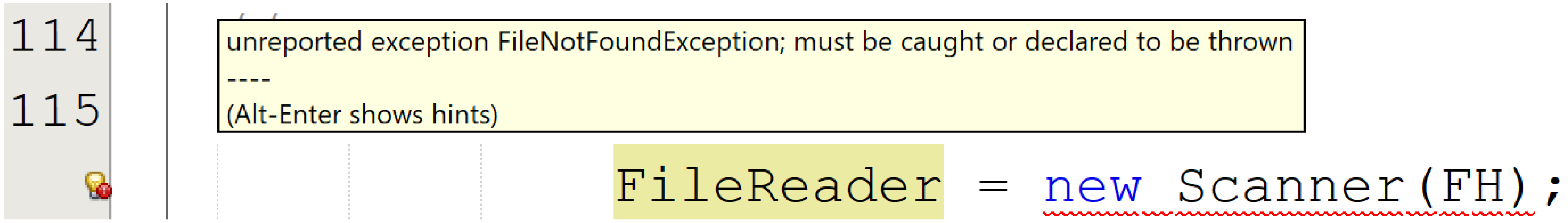
In this particular case, adding "`throws ArithmeticException`" is optional because `ArithmeticException` is a runtime error which is an unchecked error.

When dealing with checked errors, adding that `throws` is not optional.

You can see this in Coding Assignment 2

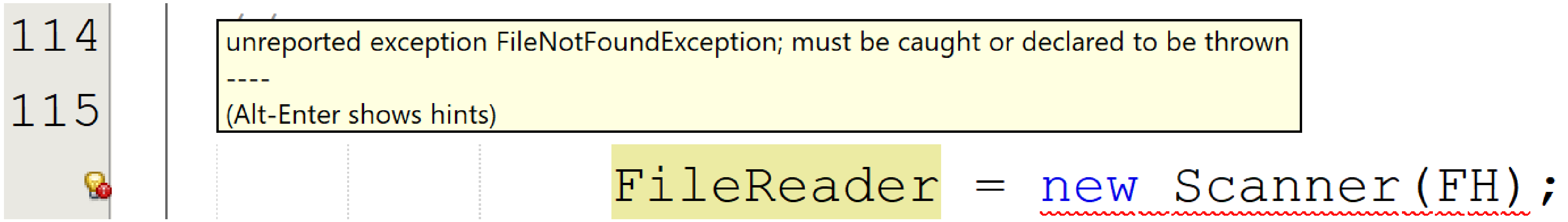
If we comment out the try-catch code, we added around our file open…

```
public static void ReadFile(String filename, int[] Levels, ArrayList<String> Colors)
{
     File FH  = new File(filename);
     Scanner FileReader = null;

//   try
//   {
          FileReader = new Scanner(FH);
//   }
//   catch (Exception e)
//   {
//        System.out.printf("%s file name does not exist...exiting\n", filename);
//        System.exit(0);
//   }
```



```
unreported exception FileNotFoundException; must be caught or declared to be thrown
----
(Alt-Enter shows hints)
```

```
FileReader = new Scanner(FH);
```

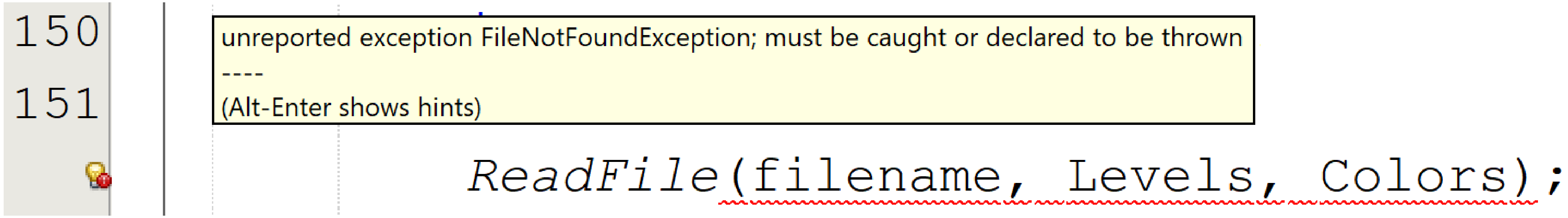


```
unreported exception FileNotFoundException; must be caught or declared to be thrown
----
(Alt-Enter shows hints)
```

```
FileReader = new Scanner(FH);
```

```
public static void ReadFile(String filename, int[] Levels, ArrayList<String> Colors)
    throws FileNotFoundException
```

Adding that fixes the error on line 116, but now we have an error in `main()`.



```
unreported exception FileNotFoundException; must be caught or declared to be thrown
----
(Alt-Enter shows hints)
```

```
ReadFile(filename, Levels, Colors);
```

We can add `FileNotFoundException` to `main()` as well.

```
public static void main(String[] args) throws FileNotFoundException
```

Now we don't have any errors BUT now what happens if we try to read a file that does not exist?

```
Exception in thread "main" java.io.FileNotFoundException: dog.catx (The
system cannot find the file specified)
	at java.base/java.io.FileInputStream.open0(Native Method)
	at java.base/java.io.FileInputStream.open(FileInputStream.java:212)
	at java.base/java.io.FileInputStream.<init>(FileInputStream.java:154)
	at java.base/java.util.Scanner.<init>(Scanner.java:639)
	at
code2_1000074079.Code2_1000074079.ReadFile(Code2_1000074079.java:117)
	at code2_1000074079.Code2_1000074079.main(Code2_1000074079.java:152)
C:\Users\frenc\Documents\NetBeansProjects\Code2_1000074079\nbproject\build-
impl.xml:1355: The following error occurred while executing this line:
C:\Users\frenc\Documents\NetBeansProjects\Code2_1000074079\nbproject\build-
impl.xml:961: Java returned: 1
```

Exception Handling

So what happens when `main()` throws an exception?

The JVM catches it but does not HANDLE it. It terminates the program.

The JVM creates a stack trace and outputs the exception message and the stack trace.

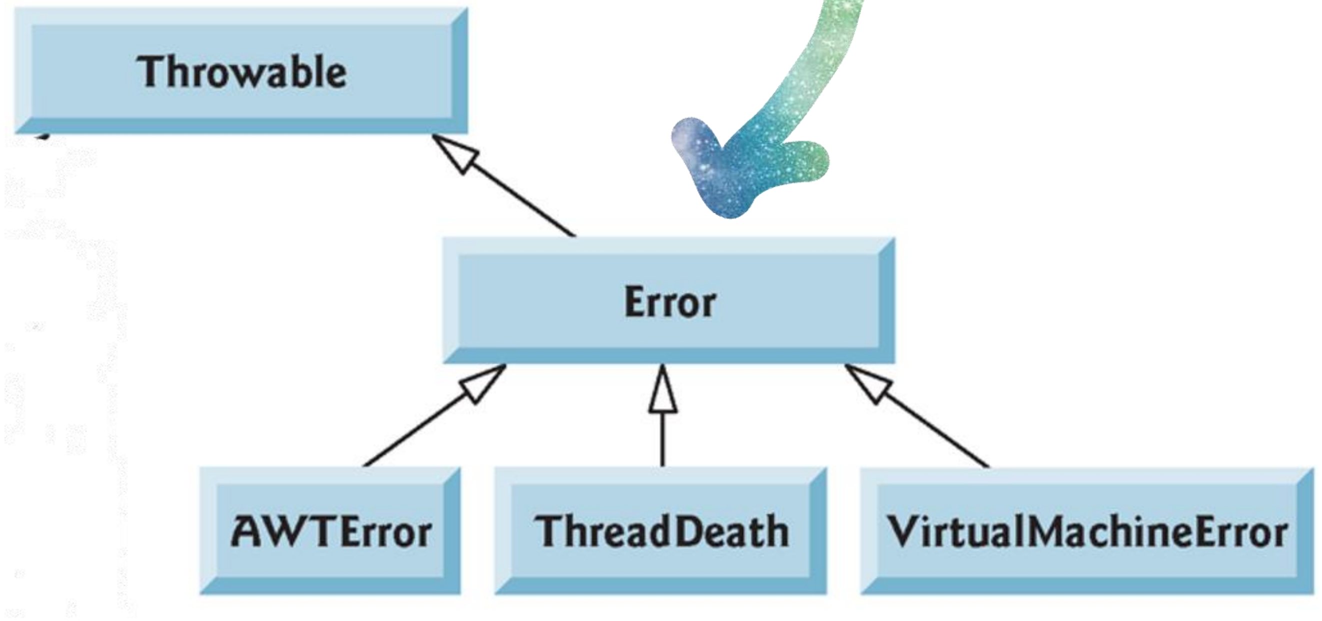
You should try as much as possible to handle your errors.

Exceptions

`Error`

Classes that inherit directly or indirectly from class `Error` are **unchecked**

`Errors` are serious problems that your program should not even attempt to deal with

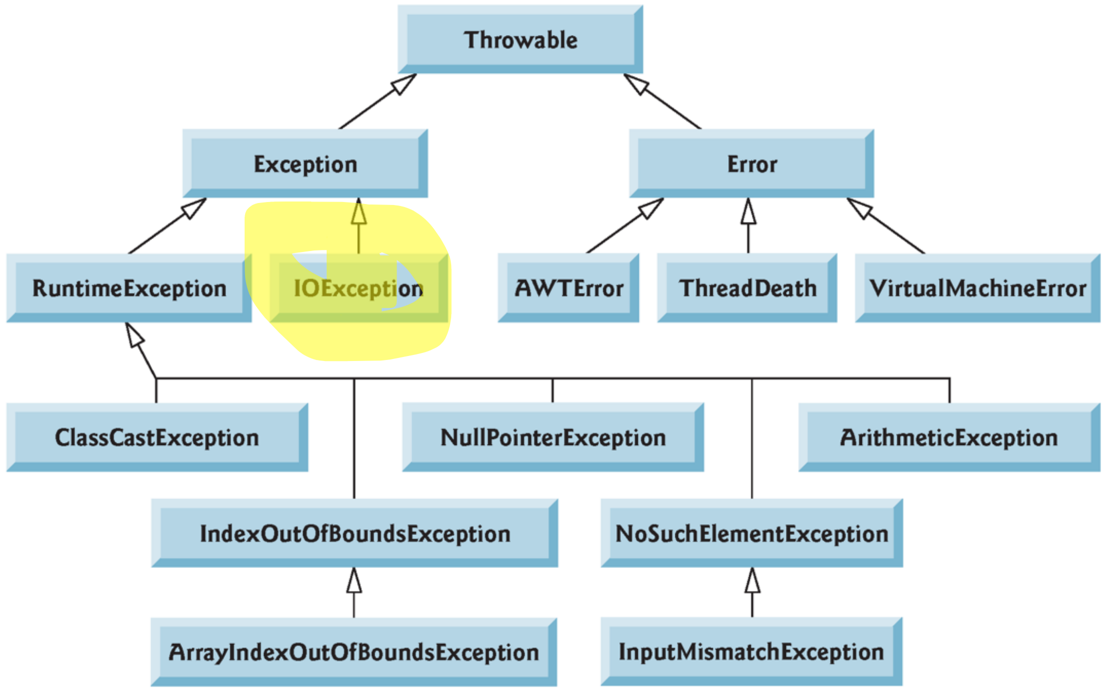


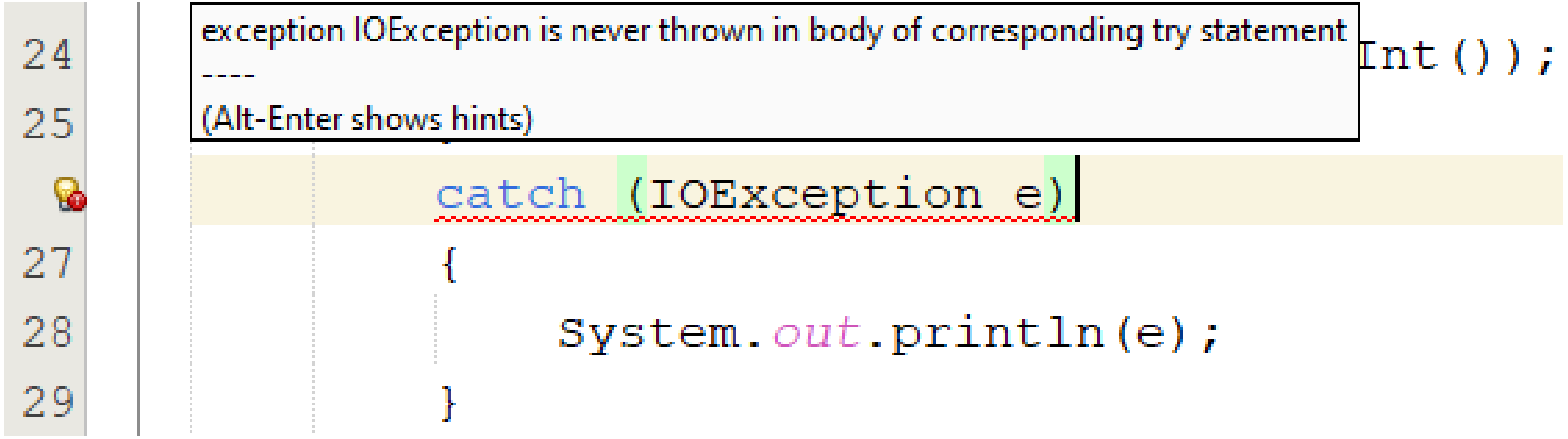
Checked Exceptions

The compiler checks each method call and method declaration to determine whether or not the method throws a checked exception.

If a method does throw a checked exception, the compiler verifies that the checked exception is caught or is declared in a `throws` clause.

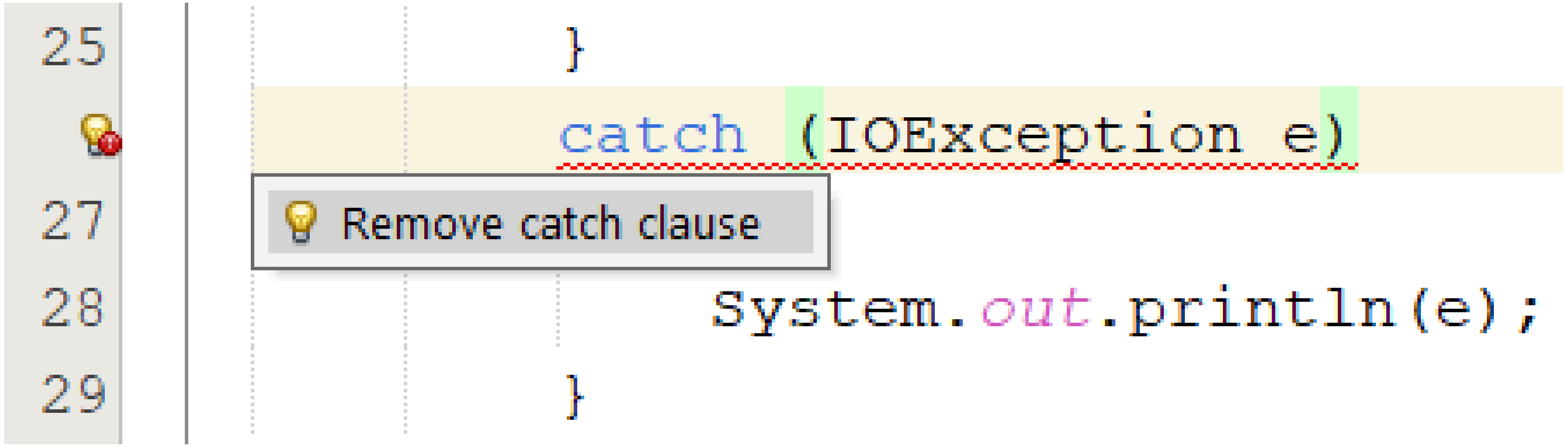
This is known as the catch-or-declare requirement





exception IOException is never thrown in body of corresponding try statement
----
(Alt-Enter shows hints)

```
Int());

        catch (IOException e)
        {
            System.out.println(e);
        }
```



```
        }
        catch (IOException e)

            System.out.println(e);
        }
```

Checked vs Unchecked Exceptions

The compiler does not enforce the *catch-or-declare* requirement for unchecked exceptions.

We were not required to put `try-catch` around `nextInt()` because it throws an `InputMismatchException` which is a subclass of `RuntimeException`.

We were required to handle the exception from opening a file that does not exist because `FileNotFoundException` is not a subclass of `RuntimeException`.

How do we know?

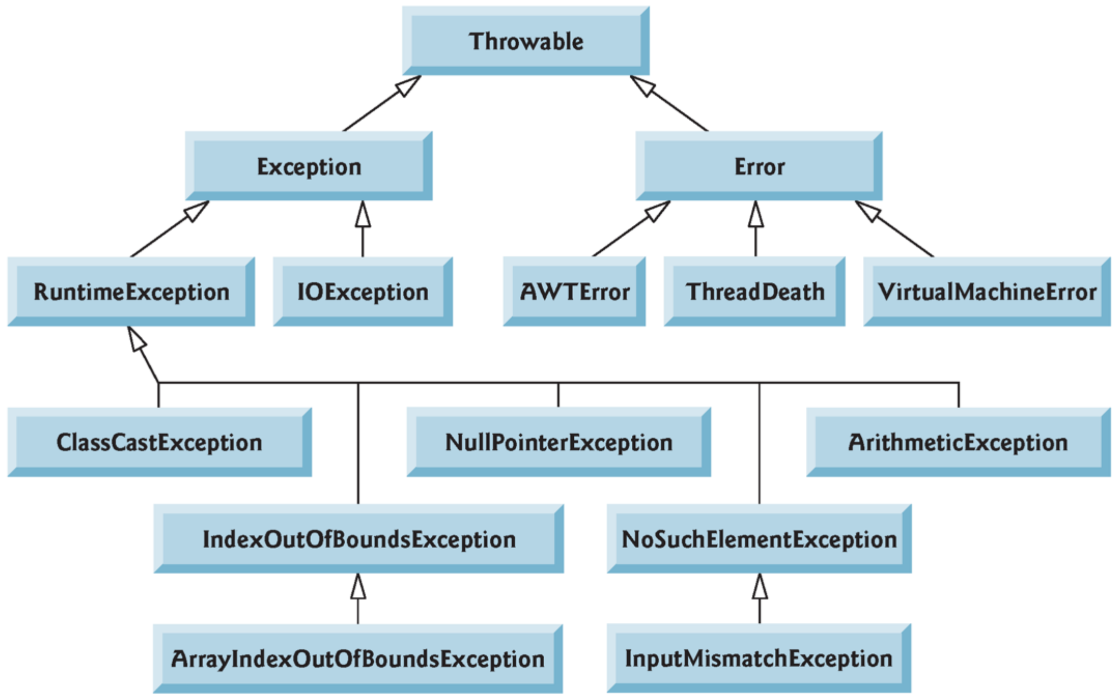
Checked vs Unchecked Exceptions

Class FileNotFoundException

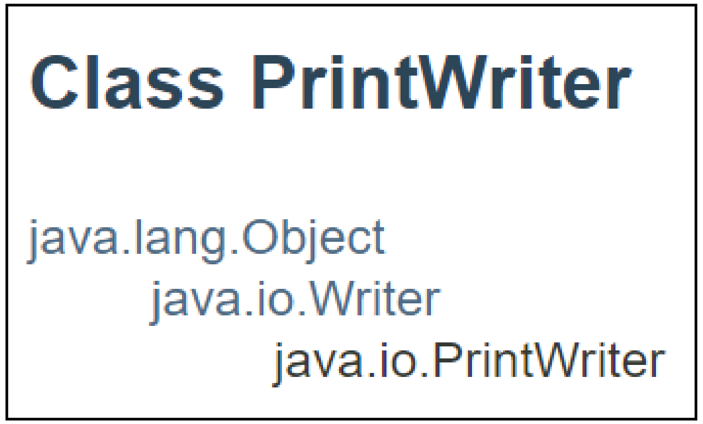
java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.io.IOException
                java.io.FileNotFoundException

Class InputMismatchException

java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.RuntimeException
                java.util.NoSuchElementException
                    java.util.InputMismatchException



```
public static void main(String[] args)
{
    PrintWriter out = new PrintWriter("output.txt");
    Scanner in = new Scanner(System.in);

    System.out.print("Enter first number ");
    int number = in.nextInt();
    out.printf("%d\n", number);

    System.out.print("Enter second number ");
    number = in.nextInt();
    out.printf("%d\n", number);

    out.close();
}
```



Class PrintWriter

```
java.lang.Object
    java.io.Writer
        java.io.PrintWriter
```

PrintWriter

```
public PrintWriter(String fileName)
            throws FileNotFoundException
```

Creates a new PrintWriter, without automatic line flushing, with the specified file name. This convenience constructor creates the necessary intermediate `OutputStreamWriter`, which will encode characters using the default charset for this instance of the Java virtual machine.

Parameters:

`fileName` - The name of the file to use as the destination of this writer. If the file exists then it will be truncated to zero size; otherwise, a new file will be created. The output will be written to the file and is buffered.

Throws:

`FileNotFoundException` - If the given string does not denote an existing, writable regular file and a new regular file of that name cannot be created, or if some other error occurs while opening or creating the file

`SecurityException` - If a security manager is present and `checkWrite(fileName)` denies write access to the file

Since:

1.5

Class FileNotFoundException

```
java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.IOException
                java.io.FileNotFoundException
```

All Implemented Interfaces:

`Serializable`

```
public class FileNotFoundException
extends IOException
```

Signals that an attempt to open the file denoted by a specified pathname has failed.

This exception will be thrown by the `FileInputStream`, `FileOutputStream`, and `RandomAccessFile` constructors when a file with the specified pathname does not exist. It will also be thrown by these constructors if the file does exist but for some reason is inaccessible, for example when an attempt is made to open a read-only file for writing.

Class Exception

```
java.lang.Object
    java.lang.Throwable
        java.lang.Exception
```

All Implemented Inter Inheritance Tree

`Serializable`

Direct Known Subclasses:

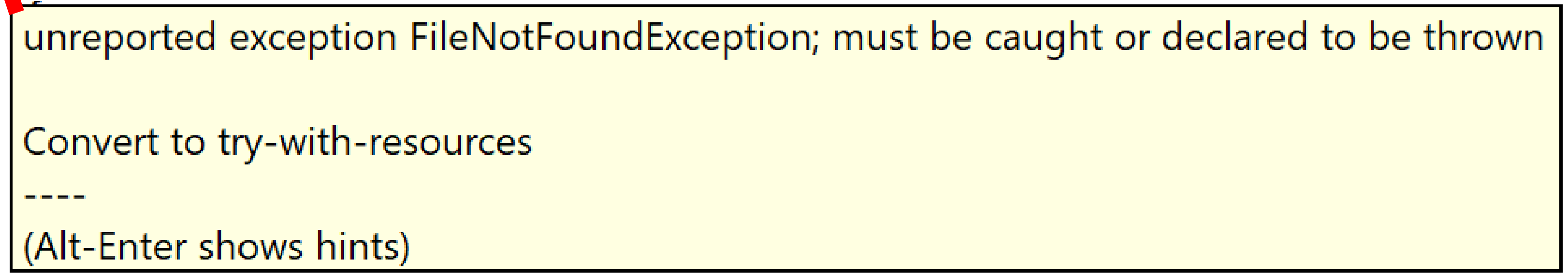
`AbsentInformationException, ActivationException, AgentInitializationException, AgentLoadException, AlreadyBoundException, AttachNotSupportedException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperation`
`BrokenBarrierException, CardException, CertificateException, ClassNotLoadedException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionControl.ExecutionControlException, ExecutionException, ExpandVetoException, FontFormatExc`
`GeneralSecurityException, GSSException, IllegalClassFormatException, IllegalConnectorArgumentsException, IncompatibleThreadStateException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException,`
`InvalidTargetObjectTypeException, InvalidTypeException, InvocationException, IOException, JMException, JShellException, KeySelectorException, LambdaConversionException, LineUnavailableException, MarshalException, MidiUnavailableException, MimeTypeParseException, NamingException,`
`NoninvertibleTransformException, NotBoundException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, ReflectiveOperationException, RefreshFailedException, RuntimeException, SAXException, ScriptException,`
`ServerNotActiveException, SQLException, StringConcatException, TimeoutException, TooManyListenersException, TransformerException, TransformException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeel`
`URIReferenceException, URISyntaxException, VMStartException, XAException, XMLParseException, XMLSignatureException, XMLStreamException, XPathException`

```
public class Exception
extends Throwable
```

The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.

The class `Exception` and any subclasses that are not also subclasses of `RuntimeException` are *checked exceptions*. Checked exceptions need to be declared in a method or constructor's `throws` clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

```
public static void main(String[] args)
{
    PrintWriter out = new PrintWriter("output.txt");
    Scanner in = new Scanner(System.in);

    System.out.print("Enter first number ");
    int number = in.nextInt();
    out.printf("%d\n", number);

    System.out.print("Enter second number ");
    number = in.nextInt();
    out.printf("%d\n", number);

    out.close();
}
```



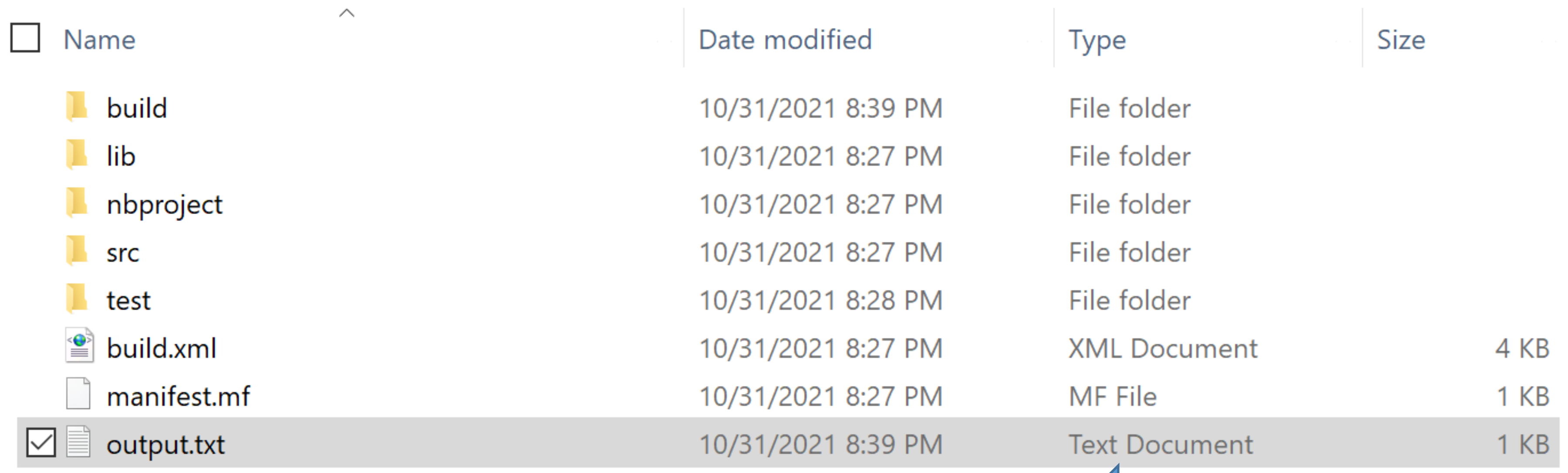
unreported exception FileNotFoundException; must be caught or declared to be thrown

Convert to try-with-resources

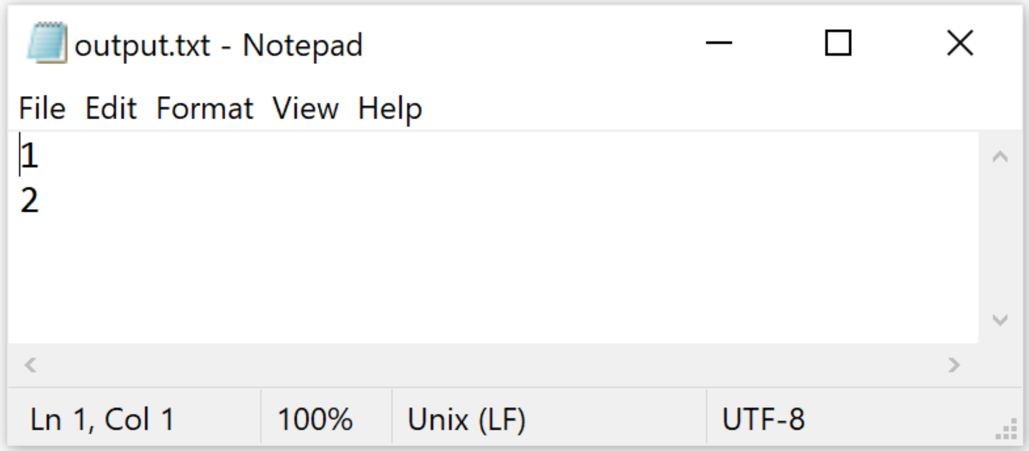
(Alt-Enter shows hints)

```java
public static void main(String[] args) throws FileNotFoundException
{
    PrintWriter out = new PrintWriter("output.txt");
    Scanner in = new Scanner(System.in);

    System.out.print("Enter first number ");
    int number = in.nextInt();
    out.printf("%d\n", number);

    System.out.print("Enter second number ");
    number = in.nextInt();
    out.printf("%d\n", number);

    out.close();
}
```

```
Enter first number 1
Enter second number 2
```



Name	Date modified	Type	Size
build	10/31/2021 8:39 PM	File folder	
lib	10/31/2021 8:27 PM	File folder	
nbproject	10/31/2021 8:27 PM	File folder	
src	10/31/2021 8:27 PM	File folder	
test	10/31/2021 8:28 PM	File folder	
build.xml	10/31/2021 8:27 PM	XML Document	4 KB
manifest.mf	10/31/2021 8:27 PM	MF File	1 KB
output.txt	10/31/2021 8:39 PM	Text Document	1 KB



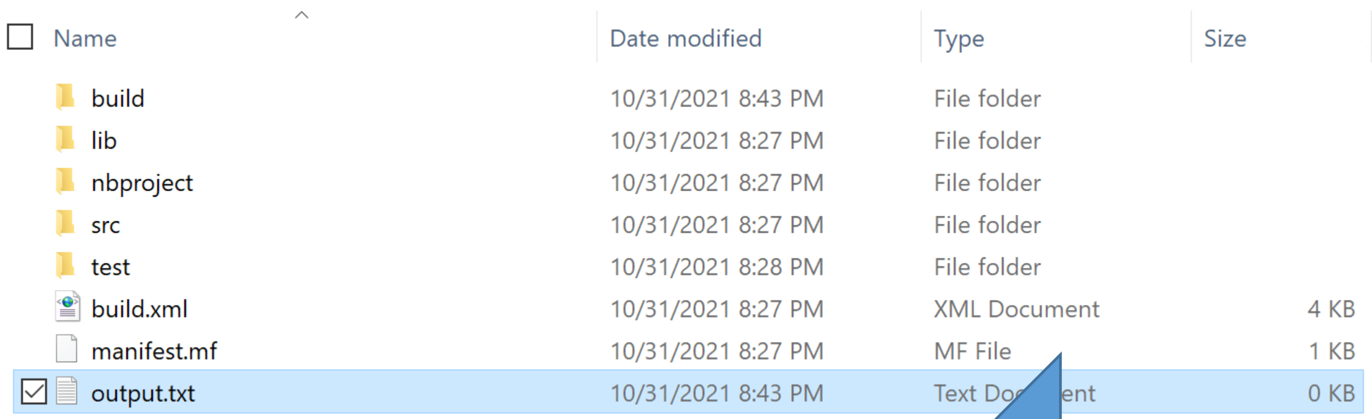
```
1
2
```

File did not exist so it was created.

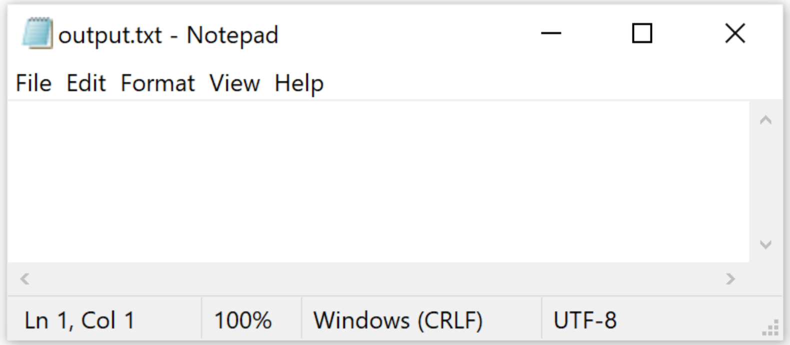
```
//          out.close();
```

What happens if we don't have the `close`?

```
Enter first number 1
Enter second number 2
```

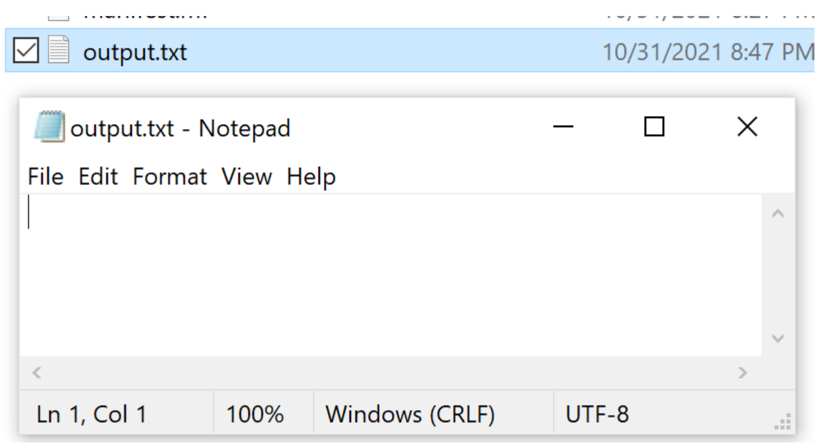


Name	Date modified	Type	Size
build	10/31/2021 8:43 PM	File folder	
lib	10/31/2021 8:27 PM	File folder	
nbproject	10/31/2021 8:27 PM	File folder	
src	10/31/2021 8:27 PM	File folder	
test	10/31/2021 8:28 PM	File folder	
build.xml	10/31/2021 8:27 PM	XML Document	4 KB
manifest.mf	10/31/2021 8:27 PM	MF File	1 KB
output.txt	10/31/2021 8:43 PM	Text Document	0 KB



We did not close the file so when the program ended, the buffer was not written to the file and our output was lost.

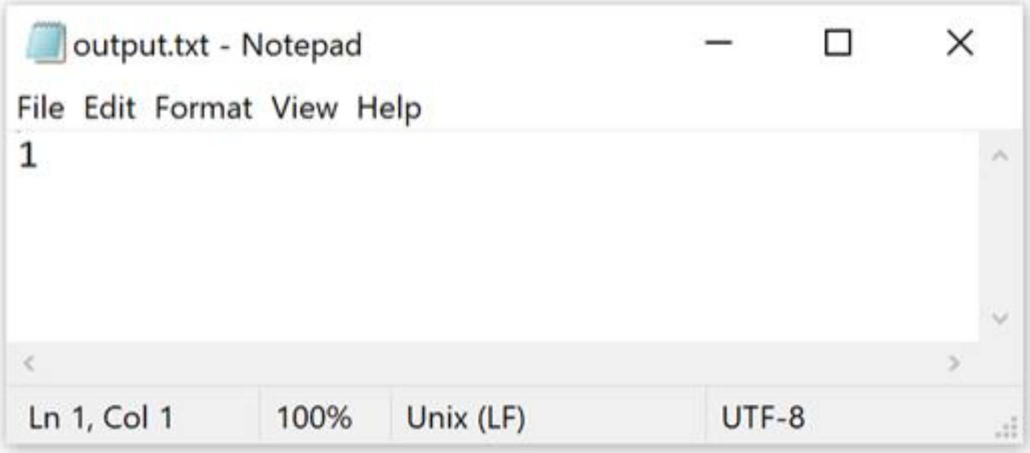
```
Enter first number 1
Enter second number a
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at finallydemo.FinallyDemo.main(FinallyDemo.java:24)
C:\Users\frenc\Documents\NetBeansProjects\FinallyDemo\nbproject\build-impl.xml:1355: The following error occurred while executing this line:
C:\Users\frenc\Documents\NetBeansProjects\FinallyDemo\nbproject\build-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 5 seconds)
```


output.txt 10/31/2021 8:47 PM

If `nextInt()` throws an exception and we do not catch it, the program ends and the `close()` is not executed.

```
PrintWriter out = new PrintWriter("output.txt");
Scanner in = new Scanner(System.in);

System.out.print("Enter first number ");

try
{
    out.printf("%d\n", in.nextInt());
    System.out.print("Enter second number ");
    out.printf("%d\n", in.nextInt());
}
catch (Exception e)
{
    System.out.println(e);
}

out.close();
```

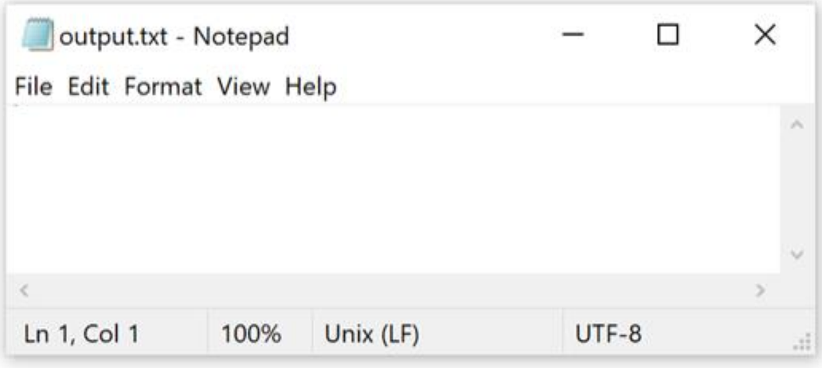
```
Enter first number 1
Enter second number a

java.util.InputMismatchException
```

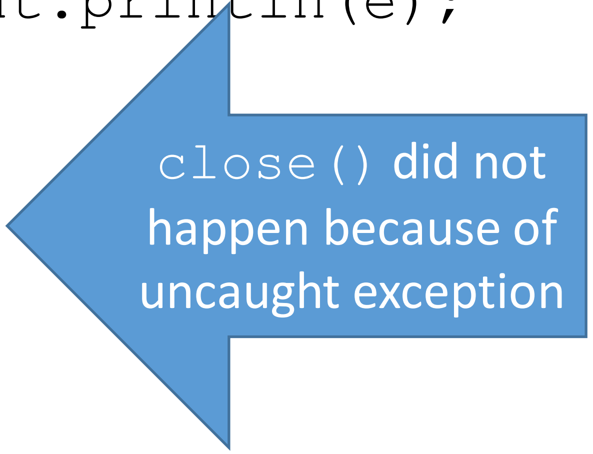


```
1
```

```
PrintWriter out = new PrintWriter("output.txt");
Scanner in = new Scanner(System.in);

System.out.print("Enter first number ");
try
{
    out.printf("%d\n", in.nextInt());
    System.out.print("Enter second number ");
    out.printf("%d\n", in.nextInt());
}
catch (NullPointerException e)
{
    System.out.println(e);
}

out.close();
```

```
Enter first number 1
Enter second number a
```

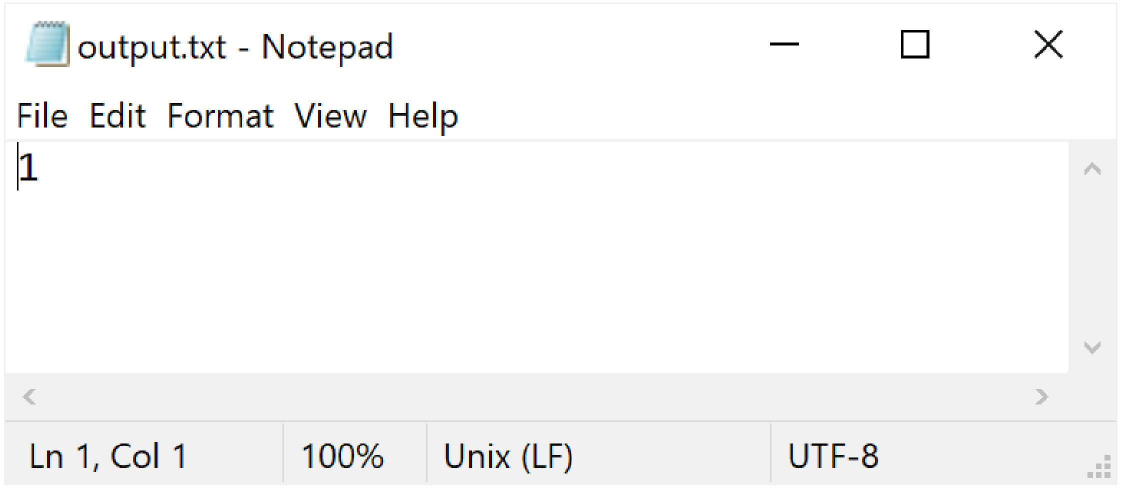




```
Enter first number 1
Enter second number a
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:943)
        at java.base/java.util.Scanner.next(Scanner.java:1598)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2263)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2217)
        at studentcode.StudentCode.main(StudentCode.java:24)
C:\Users\Donna\AppData\Local\NetBeans\Cache\14\executor-snippets\run.xml:111:
 The following error occurred while executing this line:
C:\Users\Donna\AppData\Local\NetBeans\Cache\14\executor-snippets\run.xml:68:
Java returned: 1
BUILD FAILED (total time: 22 seconds)
```

```java
PrintWriter out = new PrintWriter("output.txt");
Scanner in = new Scanner(System.in);

System.out.print("Enter first number ");
try
{
    out.printf("%d\n", in.nextInt());
    System.out.print("Enter second number ");
    out.printf("%d\n", in.nextInt());
}
catch (NullPointerException e)
{
    System.out.println(e);
}
finally
{
    out.close();
}
```

```
Enter first number 1
Enter second number a

java.util.InputMismatchException
```



```
1
```

`finally` Block

The `finally` block is optional.

If it is used, it is placed after the last `catch` block.

The `finally` block executes whether or not an exception is thrown in the `try` block.

If an exception occurs in a `try` block and is not caught by the `try`'s `catch`, the program skips the rest of the `try` block and executes the `finally` block.

The exception is passed to the next outer `try` block (or goes uncaught).

```
Scanner Apple = new Scanner(System.in);
int Pear = 0;

System.out.print("Enter a number greater than 3 : ");
Pear = Apple.nextInt();

if (Pear <= 3)
    System.out.println("HEY - THAT'S NOT GREATER THAN 3");

System.out.println(Pear);
```

```
Enter a number greater than 3 : 2
HEY - THAT'S NOT GREATER THAN 3
2
```

What if we want the program to temporarily throw an exception instead?

`assert`

When implementing and debugging a class, it's sometimes useful to state conditions that should be true at a particular point in a method.

Assertions help you debug and identify logic errors in your code.

The `assert` statement evaluates a boolean expression and, if false, throws an `AssertionError`.

`AssertionError` is a subclass of `Error`.

`Error`?
Do we catch those?

`assert`

```
assert expression;
```

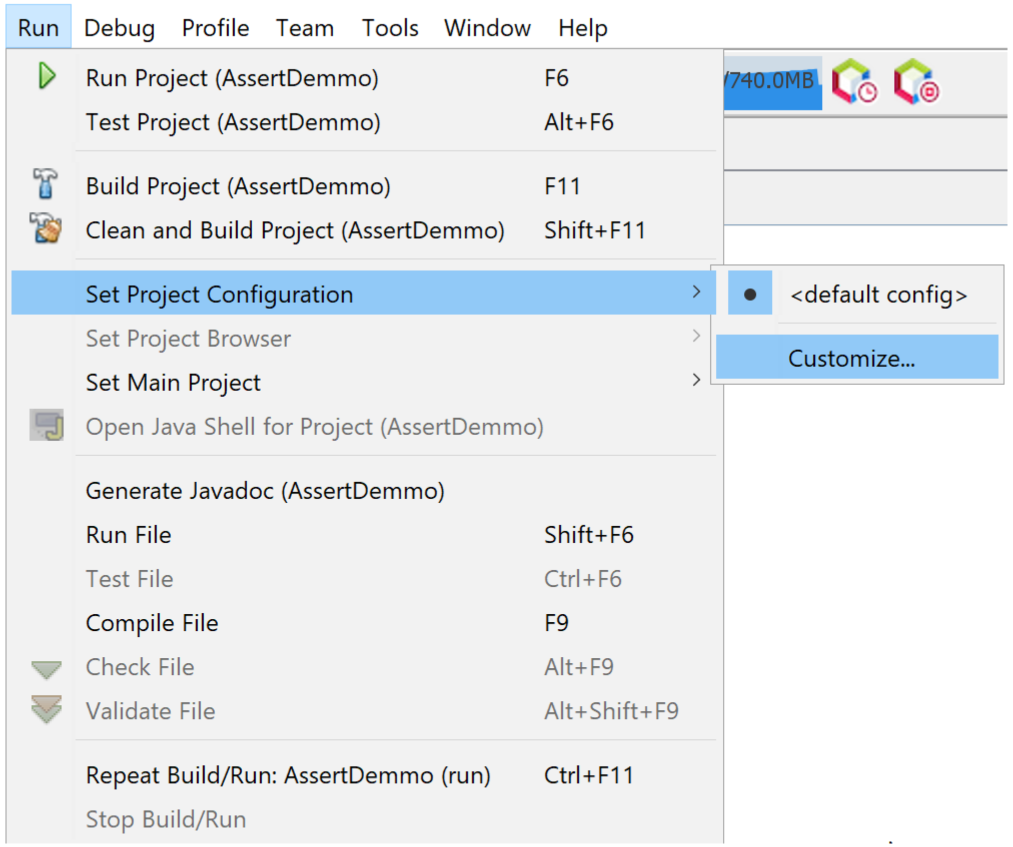
throws an `AssertionError` if expression is `false`

```
assert expression1 : expression2
```

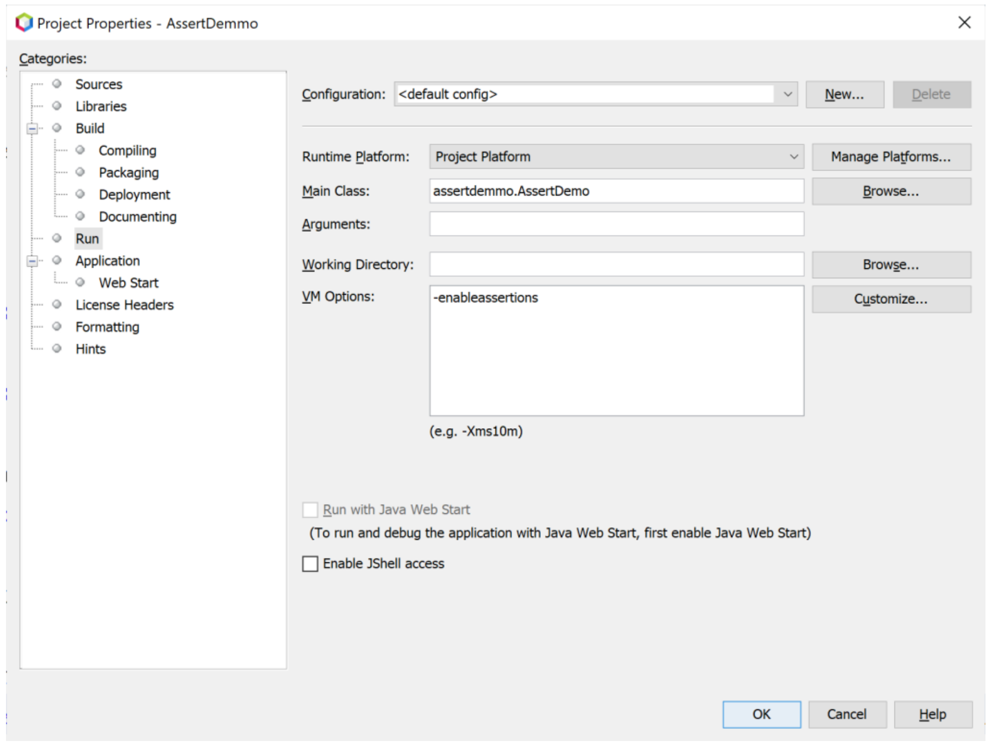
evaluates `expression1` and throws an `AssertionError` with `expression2` as the error message if `expression1` is `false`

assert

Assertions must be explicitly be enabled when executing a program because they reduce performance and they are unneeded by the user.



Run
Debug
Profile
Team
Tools
Window
Help
Run Project (AssertDemmo) F6
Test Project (AssertDemmo) Alt+F6
Build Project (AssertDemmo) F11
Clean and Build Project (AssertDemmo) Shift+F11
Set Project Configuration
Set Project Browser
Set Main Project
Open Java Shell for Project (AssertDemmo)
Generate Javadoc (AssertDemmo)
Run File Shift+F6
Test File Ctrl+F6
Compile File F9
Check File Alt+F9
Validate File Alt+Shift+F9
Repeat Build/Run: AssertDemmo (run) Ctrl+F11
Stop Build/Run
740.0MB
<default config>
Customize...


Project Properties - AssertDemmo

Categories: Sources, Libraries, Build, Compiling, Packaging, Deployment, Documenting, Run, Application, Web Start, License Headers, Formatting, Hints

Configuration: <default config>

Runtime Platform: Project Platform

Main Class: assertdemmo.AssertDemo

Arguments:

Working Directory:

VM Options: -enableassertions

(e.g. -Xms10m)

Run with Java Web Start

(To run and debug the application with Java Web Start, first enable Java Web Start)

Enable JShell access

```java
Scanner Apple = new Scanner(System.in);
int Pear = 0;

System.out.print("Enter a number greater than 3 : ");
Pear = Apple.nextInt();

assert(Pear > 3) : "HEY - THAT'S NOT GREATER THAN 3";

//if (Pear <= 3)
//    System.out.println("HEY - THAT'S NOT GREATER THAN 3");

System.out.println(Pear);
```

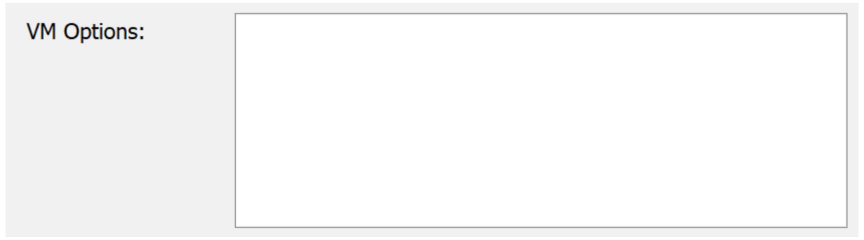
```
Enter a number greater than 3 : 2
Exception in thread "main" java.lang.AssertionError: HEY - THAT'S NOT
GREATER THAN 3
	at assertdemmo.AssertDemo.main(AssertDemo.java:18)
C:\Users\frenc\Documents\NetBeansProjects\AssertDemmo\nbproject\build-impl.xml:1355: The following error occurred while
executing this line:
C:\Users\frenc\Documents\NetBeansProjects\AssertDemmo\nbproject\build-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 3 seconds)
```

```
Scanner Apple = new Scanner(System.in);
int Pear = 0;

System.out.print("Enter a number greater than 3 : ");
Pear = Apple.nextInt();

assert(Pear > 3);

//if (Pear <= 3)
//    System.out.println("HEY - THAT'S NOT GREATER THAN 3");

System.out.println(Pear);
```

```
Enter a number greater than 3 : 2
Exception in thread "main" java.lang.AssertionError
	at assertdemmo.AssertDemo.main(AssertDemo.java:18)
C:\Users\frenc\Documents\NetBeansProjects\AssertDemmo\nbproject\build-impl.xml:1355: The following error occurred while
executing this line:
C:\Users\frenc\Documents\NetBeansProjects\AssertDemmo\nbproject\build-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 3 seconds)
```

assert

If we turn assertions off


VM Options:

```
Enter a number greater than 3 : 4
4

Enter a number greater than 3 : 2
2
```