Intro. & Learning Objectives

Method execution and exception propagation

Exception handling

Checked exceptions

Unchecked exceptions

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Material based on *Java actually: A Comprehensive Primer in Programming* (Chapter 10)

Topics Covered

1. Introduction and Learning Objectives
2. Method execution and exception prpgtion
    - Method execution
    - Stack trace
    - Exception propagation
3. Exception handling
    - **try-catch** scenario 1: no exception
    - **try-catch** scenario 2: exception
    - **try-catch** scenario 3: exc. propagation
4. Checked exceptions
    - Checked exceptions using **throw**
    - Programming with checked exceptions
5. Unchecked exceptions

What is an exception?

- A program must be able to handle error situations gracefully when they occur at runtime
- This is the role of exception handling provided by Java
- Error situations can be divided into two main categories:
  - Programming errors (e.g., an invalid index to access an array)
    Ideally, they should not occur
  - Runtime environment errors (e.g., opening a file that does not exist)
    Should be properly handled

Intro. & Learning Objectives
Method execution and exception propagation
Exception handling
Checked exceptions
Unchecked exceptions

Exception Handling
Introduction and Learning Objectives

- A program must be able to handle both kinds of errors
- An exception signals that an error or an unexpected situation has occurred during program execution
- It is based on the "throw and catch" principle
- An exception is thrown when an error situation occurs during program execution
- It is caught by an exception handler that takes an appropriate action to handle the situation

Intro. & Learning Objectives

Method execution and exception propagation

Exception handling

Checked exceptions

Unchecked exceptions

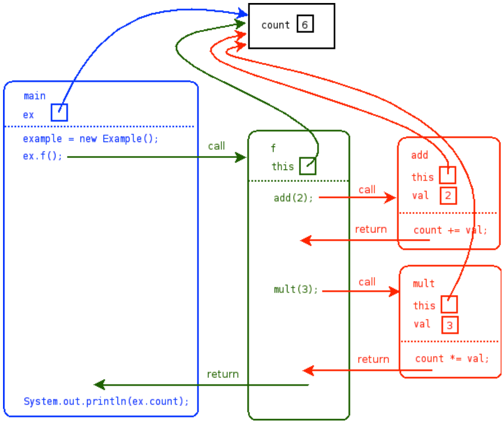
- This principle is embedded in the try-catch statement
- All exceptions are objects
- Java standard library provides classes that represent different types of exception

Learning Objectives

- Use of exception handling to create programs that are reliable and robust
- Major scenarios of program execution when using the try-catch statement
- Understand how exceptions are thrown, propagated, caught and handled
- Understand the difference between checked and unchecked exceptions

Intro. & Learning Objectives
Method execution and exception propagation
Exception handling
Checked exceptions
Unchecked exceptions

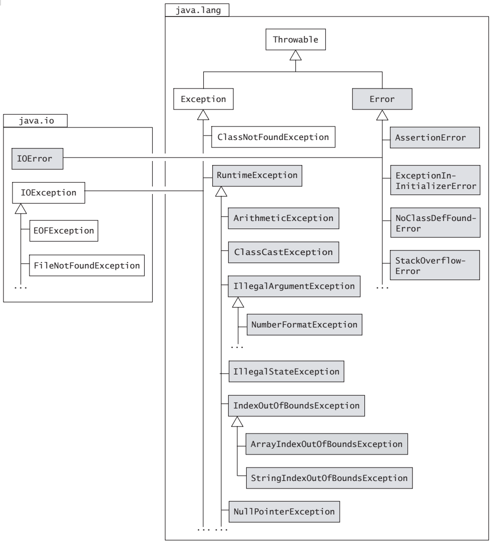
Method Execution (Normal Execution)



Program Stack to control the execution

SFWR ENG/COMP SCI 2S03 Principles of Programming
Dr. R. Khedri
Intro. & Learning Objectives
Method execution and exception propagation
Method execution
Stack trace
Exception propagation
Exception handling
Checked exceptions
Unchecked exceptions

Exception Handling
Method execution and exception prpgtion
Method execution



Classes that are shaded (and their subclasses) represent unchecked exceptions.

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Method execution and exception propagation

Method execution

Stack trace

Exception propagation

Exception handling

Checked exceptions

Unchecked exceptions

Exception Handling

Method execution and exception prpgtion

Method execution

```
public class Speed1 {

  public static void main(String[] args) {
    System.out.println("Entering main().");
    printSpeed(100, 20);                          // (1)
    System.out.println("Returning from main().");
  }

  private static void printSpeed(int kilometers, int hours) {
    System.out.println("Entering printSpeed().");
    int speed = calculateSpeed(kilometers, hours);    // (2)
    System.out.println("Speed = " +
                     kilometers + "/" + hours + " = " + speed);
    System.out.println("Returning from printSpeed().");
  }

  private static int calculateSpeed(int distance, int time) {
    System.out.println("Calculating speed.");
    return distance/time;                         // (3)
  }
}
```

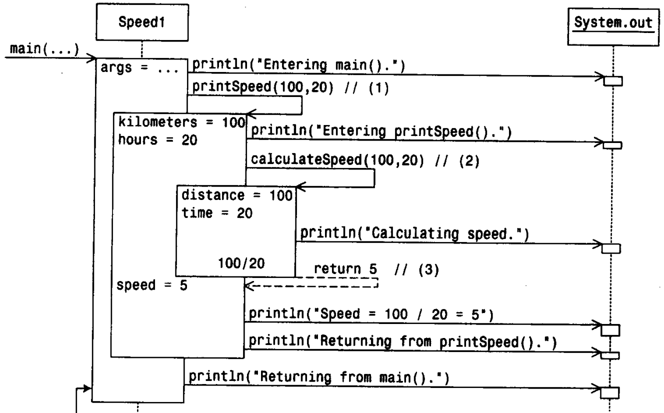
```
Entering main().
Entering printSpeed().
Calculating speed.
Speed = 100/20 = 5
Returning from printSpeed().
Returning from main().
```

Intro. & Learning Objectives
Method execution and exception propagation
Method execution
Stack trace
Exception propagation
Exception handling
Checked exceptions
Unchecked exceptions

Exception Handling

Method execution and exception prpgtion

Method execution


Speed1
System.out
main(...)
args = ...
println("Entering main().")
printSpeed(100,20) // (1)
kilometers = 100
hours = 20
println("Entering printSpeed().")
calculateSpeed(100,20) // (2)
distance = 100
time = 20
println("Calculating speed.")
100/20
return 5 // (3)
speed = 5
println("Speed = 100 / 20 = 5")
println("Returning from printSpeed().")
println("Returning from main().")

Method execution

Program output:

```
Entering main().
Entering printSpeed().
Calculating speed.
Speed = 100/20 = 5
Returning from printSpeed().
Returning from main().
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Method execution and exception propagation

Method execution

Stack trace

Exception propagation

Exception handling

Checked exceptions

Unchecked exceptions

Exception Handling

Method execution and exception prpgtion

Stack trace

```java
public class Speed1WithException {

  public static void main(String[] args) {
    System.out.println("Entering main().");
    printSpeed(100, 0);                // (1) ***** CHANGED
    System.out.println("Returning from main().");
  }

  private static void printSpeed(int kilometers, int hours) {
    System.out.println("Entering printSpeed().");
    int speed = calculateSpeed(kilometers, hours);     // (2)
    System.out.println("Speed = " +
                       kilometers + "/" + hours + " = " + speed);
    System.out.println("Returning from printSpeed().");
  }

  private static int calculateSpeed(int distance, int time) {
    System.out.println("Calculating speed.");
    return distance/time;                              // (3)
  }
}
```

```
Entering main().
Entering printSpeed().
Calculating speed.
Exception in thread "main" java.lang.ArithmeticException: / by zero
  at Speed1WithException.calculateSpeed(Speed1WithException.java:20)
  at Speed1WithException.printSpeed(Speed1WithException.java:12)
  at Speed1WithException.main(Speed1WithException.java:6)
```

Intro. & Learning Objectives

Method execution and exception propagation
- Method execution
- Stack trace
- Exception propagation

Exception handling

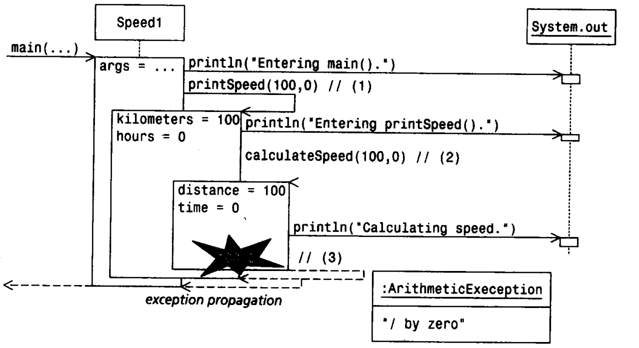
Checked exceptions

Unchecked exceptions

Exception Handling

Method execution and exception prpgtion

Stack trace



Program output: *Printed by the program* *Printed by the default exception handler*

```
Entering main().
Entering printSpeed().
Calculating speed.
Exception in thread "main" java.lang.ArithmeticException: / by zero
  at Speed1.calculateSpeed(Speed1.java:20)
  at Speed1.printSpeed(Speed1.java:12)
  at Speed1.main(Speed1.java:6)
```

class name *method name* *file name* *line number in the file where the call to the next method occurred.*

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Method execution and exception propagation

Method execution

Stack trace

Exception propagation

Exception handling

Checked exceptions

Unchecked exceptions

Intro. & Learning Objectives
Method execution and exception propagation
Method execution
Stack trace
Exception propagation
Exception handling
Checked exceptions
Unchecked exceptions

Propagation of an exception:

- The exception is offered to the method in which the exception occurred
  - No code to deal with this exception, it is terminated
  - Its stack frame is removed
- The exception is next offered to the method at the top of the program stack
  - No code to handle the exception, it is terminated
  - It is also terminated and its stack frame removed
- . . .
- The exception is next offered to the main( ) method $\leadsto$ terminated & its stack frame removed

- After the main terminated, the exception has propagated to the top level
- It is now handled by a default exception handler in the JVM
- The default exception handler prints information about
  - the exception
  - the stack trace at the time when the exception occurred
- The execution of the program is then terminated (For terminal-based applications)
- The program continues (For applications with a GUI)

SFWR ENG/COMP SCI 2S03 Principles of Programming

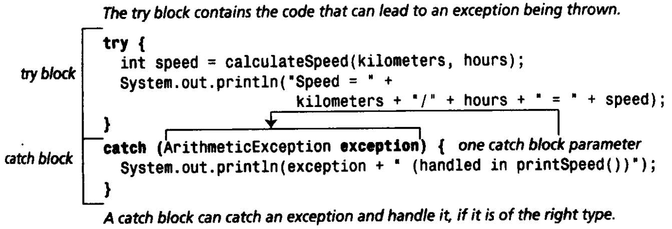
Dr. R. Khedri

Intro. & Learning Objectives
Method execution and exception propagation
Method execution
Stack trace
Exception propagation
Exception handling
Checked exceptions
Unchecked exceptions

- Not advisable to leave exceptions to a default handler
- The consequences:
  - data can be lost
  - lost of control of the system
- The construct **try-catch** can be used for exception handling in Java
- It is formed by a try block followed by a catch block
- A try block contains statements that can result in an exception being thrown during execution
- A catch block constitutes an exception handler

Intro. & Learning Objectives
Method execution and exception propagation
Exception handling
try-catch scenario 1: no exception
try-catch scenario 2: exception handling
try-catch scenario 3: exception propagation
Checked exceptions
Unchecked exceptions

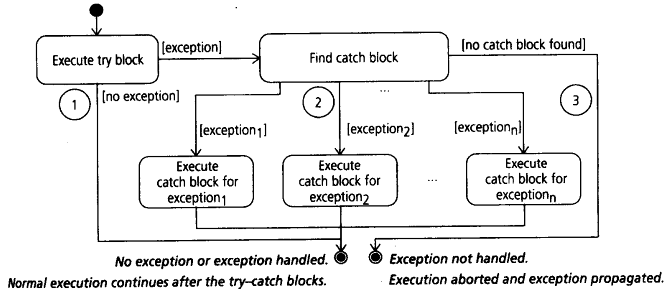
Exception Handling
Exception handling



The try block contains the code that can lead to an exception being thrown.

```
try {
  int speed = calculateSpeed(kilometers, hours);
  System.out.println("Speed = " +
                     kilometers + "/" + hours + " = " + speed);
}
catch (ArithmeticException exception) {
  System.out.println(exception + " (handled in printSpeed())");
}
```

A catch block can catch an exception and handle it, if it is of the right type.


Execute try block
[exception]
Find catch block
[no catch block found]
[no exception]
1
2
3
...
[exception$_1$]
[exception$_2$]
[exception$_n$]
Execute catch block for exception$_1$
Execute catch block for exception$_2$
...
Execute catch block for exception$_n$
No exception or exception handled.
Normal execution continues after the try-catch blocks.
Exception not handled.
Execution aborted and exception propagated.

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Method execution and exception propagation

Exception handling
try-catch scenario 1: no exception
try-catch scenario 2: exception handling
try-catch scenario 3: exception propagation

Checked exceptions

Unchecked exceptions

Exception Handling

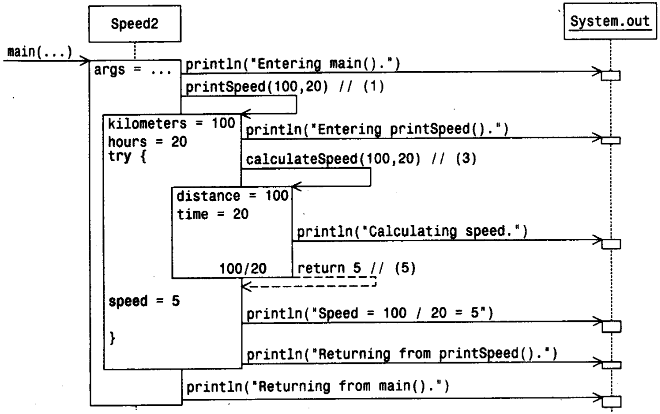
Exception handling

try-catch scenario 1: no exception

```
public class Speed2 {

  public static void main(String[] args) {
    System.out.println("Entering main().");
    printSpeed(100, 20);                                        // (1)
    System.out.println("Returning from main().");
  }

  private static void printSpeed(int kilometers, int hours) {
    System.out.println("Entering printSpeed().");
    try {                                                      // (2)
      int speed = calculateSpeed(kilometers, hours);            // (3)
      System.out.println("Speed = " +
                         kilometers + "/" + hours + " = " + speed);
    }
    catch (ArithmeticException exception) {                    // (4)
      System.out.println(exception + " (handled in printSpeed())");
    }
    System.out.println("Returning from printSpeed().");
  }

  private static int calculateSpeed(int distance, int time) {
    System.out.println("Calculating speed.");
    return distance/time;                                      // (5)
  }
}
```

```
Entering main().
Entering printSpeed().
Calculating speed.
Speed = 100/20 = 5
Returning from printSpeed().
Returning from main().
```

Exception Handling
Exception handling
try-catch scenario 1: no exception


Speed2
System.out
main(...)
args = ...
println("Entering main().")
printSpeed(100,20) // (1)
kilometers = 100
hours = 20
try {
println("Entering printSpeed().")
calculateSpeed(100,20) // (3)
distance = 100
time = 20
println("Calculating speed.")
100/20
return 5 // (5)
speed = 5
println("Speed = 100 / 20 = 5")
}
println("Returning from printSpeed().")
println("Returning from main().")

Program output:
```
Entering main().
Entering printSpeed().
Calculating speed.
Speed = 100/20 = 5
Returning from printSpeed().
Returning from main().
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Method execution and exception propagation

Exception handling
- try-catch scenario 1: no exception
- try-catch scenario 2: exception handling
- try-catch scenario 3: exception propagation

Checked exceptions

Unchecked exceptions

Exception Handling
Exception handling
try-catch scenario 2: exception

```java
public class Speed2WithException {

  public static void main(String[] args) {
    System.out.println("Entering main().");
    printSpeed(100, 0);                     // (1) **** ERROR HERE
    System.out.println("Returning from main().");
  }

  private static void printSpeed(int kilometers, int hours) {
    System.out.println("Entering printSpeed().");
    try {                                                     // (2)
      int speed = calculateSpeed(kilometers, hours);          // (3)
      System.out.println("Speed = " +
                         kilometers + "/" + hours + " = " + speed);
    }
    catch (ArithmeticException exception) {                   // (4)
      System.out.println(exception + " (handled in printSpeed())");
    }
    System.out.println("Returning from printSpeed().");
  }

  private static int calculateSpeed(int distance, int time) {
    System.out.println("Calculating speed.");
    return distance/time;                                     // (5)
  }
}
```

```
Entering main().
Entering printSpeed().
Calculating speed.
java.lang.ArithmeticException: / by zero (handled in printSpeed())
Returning from printSpeed().
Returning from main().
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

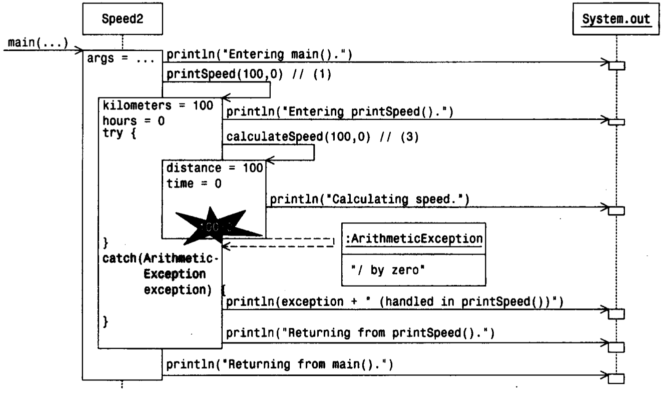
Method execution and exception propagation

Exception handling
- try-catch scenario 1: no exception
- try-catch scenario 2: exception handling
- try-catch scenario 3: exception propagation

Checked exceptions

Unchecked exceptions

Exception Handling
Exception handling
try-catch scenario 2: exception


Speed2
System.out
main(...)
args = ...
println("Entering main().")
printSpeed(100,0) // (1)
kilometers = 100
hours = 0
try {
println("Entering printSpeed().")
calculateSpeed(100,0) // (3)
distance = 100
time = 0
println("Calculating speed.")
100/0
:ArithmeticException
"/ by zero"
}
catch(Arithmetic-Exception exception) {
println(exception + " (handled in printSpeed())")
}
println("Returning from printSpeed().")
println("Returning from main().")

Program output:
```
Entering main().
Entering printSpeed().
Calculating speed.
java.lang.ArithmeticException: / by zero (handled in printSpeed())
Returning from printSpeed().
Returning from main().
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Method execution and exception propagation

Exception handling
- try-catch scenario 1: no exception
- try-catch scenario 2: exception handling
- try-catch scenario 3: exception propagation

Checked exceptions

Unchecked exceptions

Exception Handling
Exception handling
try-catch scenario 3: exc. propagation

```
public class Speed3WithException {

  public static void main(String[] args) {
    System.out.println("Entering main().");
    try {                                                        // (1)
      printSpeed(100,0);         // (2) **** ERROR
    }
    catch (ArithmeticException exception) {                      // (3)
      System.out.println(exception + " (handled in main())");
    }
    System.out.println("Returning from main().");
  }

  private static void printSpeed(int kilometers, int hours) {
    System.out.println("Entering printSpeed().");
    try {                                                        // (4)
      int speed = calculateSpeed(kilometers, hours);            //
           (5)
      System.out.println("Speed = " +
                         kilometers + "/" + hours + " = " + speed);
    }
    catch (IllegalArgumentException exception) {                 // (6)
      System.out.println(exception + " (handled in printSpeed())");
    }
    System.out.println("Returning from printSpeed().");
  }

  private static int calculateSpeed(int distance, int time) {
    System.out.println("Calculating speed.");
    return distance/time;                                        // (7)
  }
}
```

```
Entering main().
Entering printSpeed().
Calculating speed.
java.lang.ArithmeticException: / by zero (handled in main())
Returning from main().
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

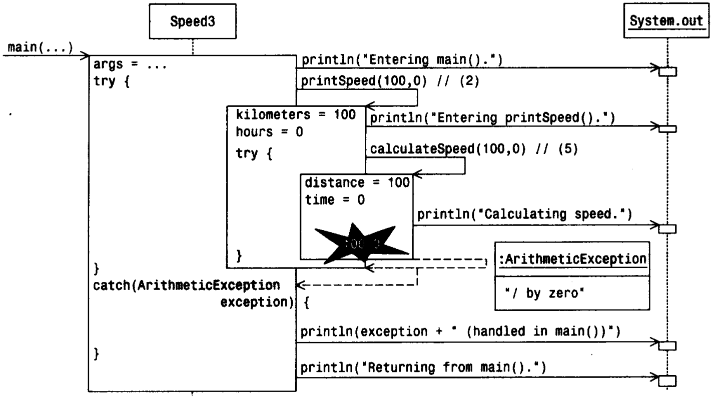
Method execution and exception propagation

Exception handling
- try-catch scenario 1: no exception
- try-catch scenario 2: exception handling
- try-catch scenario 3: exception propagation

Checked exceptions

Unchecked exceptions

try-catch scenario 3: exc. propagation


Speed3
System.out
main(...)
args = ...
try {
println("Entering main().")
printSpeed(100,0) // (2)
kilometers = 100
hours = 0
try {
println("Entering printSpeed().")
calculateSpeed(100,0) // (5)
distance = 100
time = 0
println("Calculating speed.")
}
}
:ArithmeticException
"/ by zero"
catch(ArithmeticException
exception) {
println(exception + " (handled in main())")
}
println("Returning from main().")

Program output:

```
Entering main().
Entering printSpeed().
Calculating speed.
java.lang.ArithmeticException: / by zero (handled in main())
Returning from main().
```

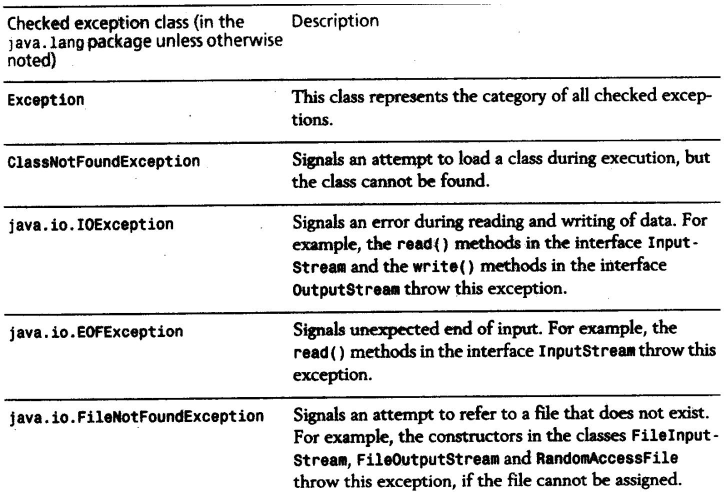
Intro. & Learning Objectives
Method execution and exception propagation
Exception handling
try-catch scenario 1: no exception
try-catch scenario 2: exception handling
try-catch scenario 3: exception propagation
Checked exceptions
Unchecked exceptions

Checked exceptions

- Java defines some special exceptions that a program cannot ignore when they are thrown
- Such an exception is called a checked exception
- The compiler will complain if the method in which it can occur does not deal with it explicitly
- Checked exceptions force the programmer to take explicit action to deal with them
- The Java standard library defines classes whose objects represent exceptions

Intro. & Learning Objectives
Method execution and exception propagation
Exception handling
Checked exceptions
Dealing with checked exceptions using the throws clause
Programming with checked exceptions
Unchecked exceptions

Exception Handling
Checked exceptions



Checked exception class (in the `java.lang` package unless otherwise noted)	Description
`Exception`	This class represents the category of all checked exceptions.
`ClassNotFoundException`	Signals an attempt to load a class during execution, but the class cannot be found.
`java.io.IOException`	Signals an error during reading and writing of data. For example, the `read()` methods in the interface `InputStream` and the `write()` methods in the interface `OutputStream` throw this exception.
`java.io.EOFException`	Signals unexpected end of input. For example, the `read()` methods in the interface `InputStream` throw this exception.
`java.io.FileNotFoundException`	Signals an attempt to refer to a file that does not exist. For example, the constructors in the classes `FileInputStream`, `FileOutputStream` and `RandomAccessFile` throw this exception, if the file cannot be assigned.

Intro. & Learning Objectives
Method execution and exception propagation
Exception handling
Checked exceptions
Dealing with checked exceptions using the throws clause
Programming with checked exceptions
Unchecked exceptions

- A method that can throw a checked exception must satisfy one of the following two conditions
  1. Catch and handle the exception in a try-catch statement
  2. Allow further propagation of the exception with a throws clause specified in its method declaration (to discuss)

- A throws clause is specified in the method header
  . . . method name (. . .) throws exception class$_1$**, . . . , exception class**$_n$ { . . . }

Programming with checked exceptions

- We can use a throw statement to throw an exception explicitly
- We specify the exception object to be thrown in the statement

```
if (distance< 0 || time <= 0)
        throw new Exception("distance and time must be > 0");
```

- We call the constructor of the exception class and pass a suitable message to explain the error situation

Intro. & Learning Objectives
Method execution and exception propagation
Exception handling
Checked exceptions
Dealing with checked exceptions using the throws clause
Programming with checked exceptions
Unchecked exceptions

Exception Handling
Checked exceptions
Programming with checked exceptions

```
public class Speed6 {

  public static void main(String[] args) {
    System.out.println("Entering main().");
    try {                                                      //(1)
    printSpeed(100, 20);                                      //(2a)
//       printSpeed(-100,20);                                    //(2b)
    }
    catch (Exception exception) {                              //(3)
      System.out.println(exception + " (handled in main())");
    }
    System.out.println("Returning from main().");
  }

  private static void printSpeed(int kilometers, int hours)
                      throws Exception {                       //(4)
    System.out.println("Entering printSpeed().");
    double speed = calculateSpeed(kilometers, hours);
    System.out.println("Speed = " +
                        kilometers + "/" + hours + " = " + speed);
    System.out.println("Returning from printSpeed().");
  }

  private static int calculateSpeed(int distance, int time)
                     throws Exception {                        //(5)
    System.out.println("Calculating speed.");
    if (distance < 0 || time <= 0)                             //(6)
      throw new Exception("distance and time must be > 0");
    return distance/time;
  }
```

```
Entering main().
Entering printSpeed().
Calculating speed.
Speed = 100/20 = 5.0
Returning from printSpeed().
Returning from main().
```

```
Entering main().
Entering printSpeed().
Calculating speed.
java.lang.Exception: distance and time must be > 0 (handled in main())
Returning from main().
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Method execution and exception propagation

Exception handling

Checked exceptions

Dealing with checked exceptions using the throws clause

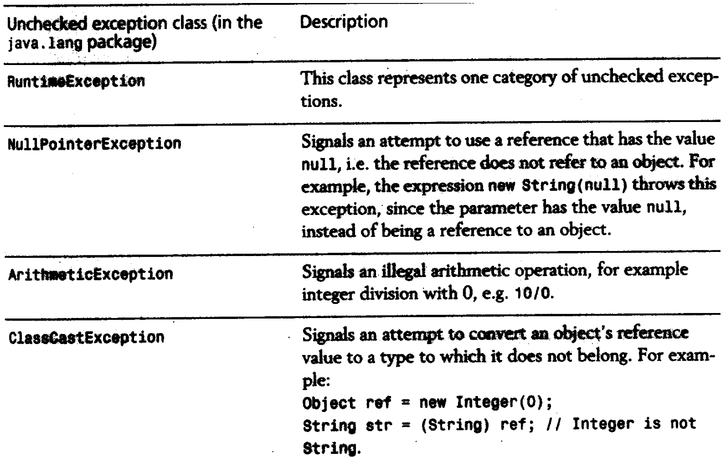
Programming with checked exceptions

Unchecked exceptions

Exception Handling
Unchecked exceptions

- Unchecked exceptions are exceptions typically concerning unforeseen errors
- The compiler DOES NOT check whether unchecked exceptions cim be thrown
- A method does not have to deal with unchecked exceptions
- The best solution for handling such situations is to correct the cause of the errors in the program(Use assertions)
- Such exceptions are not specified in the throws clause of a method

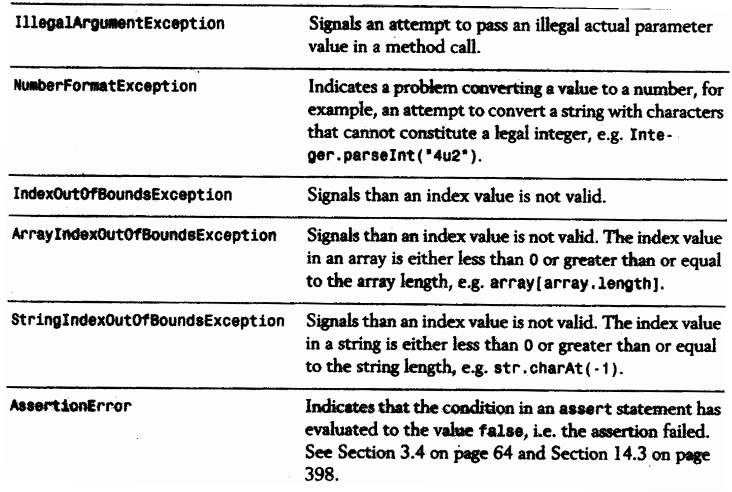
Exception Handling
Unchecked exceptions



Unchecked exception class (in the `java.lang` package)	Description
`RuntimeException`	This class represents one category of unchecked exceptions.
`NullPointerException`	Signals an attempt to use a reference that has the value `null`, i.e. the reference does not refer to an object. For example, the expression `new String(null)` throws this exception, since the parameter has the value `null`, instead of being a reference to an object.
`ArithmeticException`	Signals an illegal arithmetic operation, for example integer division with 0, e.g. `10/0`.

`ClassCastException`	Signals an attempt to convert an object's reference value to a type to which it does not belong. For example:
	`Object ref = new Integer(0);`
	`String str = (String) ref; // Integer is not String.`

Exception Handling
Unchecked exceptions



`IllegalArgumentException`	Signals an attempt to pass an illegal actual parameter value in a method call.
`NumberFormatException`	Indicates a problem converting a value to a number, for example, an attempt to convert a string with characters that cannot constitute a legal integer, e.g. `Integer.parseInt("4u2")`.
`IndexOutOfBoundsException`	Signals than an index value is not valid.
`ArrayIndexOutOfBoundsException`	Signals than an index value is not valid. The index value in an array is either less than `0` or greater than or equal to the array length, e.g. `array[array.length]`.
`StringIndexOutOfBoundsException`	Signals than an index value is not valid. The index value in a string is either less than `0` or greater than or equal to the string length, e.g. `str.charAt(-1)`.
`AssertionError`	Indicates that the condition in an `assert` statement has evaluated to the value `false`, i.e. the assertion failed. See Section 3.4 on page 64 and Section 14.3 on page 398.

Intro. & Learning Objectives

Method execution and exception propagation

Exception handling

Checked exceptions

Unchecked exceptions

Intro. & Learning Objectives

Method execution and exception propagation

Exception handling

Checked exceptions

Unchecked exceptions