4 Recursion

- **Recursion:** Divide a given problem into subproblems of the same type
  - One or more base cases
  - Rules to reduce other cases towards base case

- **Example:**
  - Factorial: 
    \[ n! = \begin{cases} 
    1 & \text{if } n = 0, \\
    (n-1)! \cdot n & \text{if } n > 0. 
    \end{cases} \]
  - Fibonacci: 
    \[ f(n) = \begin{cases} 
    1 & \text{if } n = 1, \\
    1 & \text{if } n = 2, \\
    f(n-1) + f(n-2) & \text{if } n > 2. 
    \end{cases} \]

- Deepening readings:
  - [Recursion](http://en.wikipedia.org/wiki/Recursion)
  - [Factorial](http://en.wikipedia.org/wiki/Factorial)
4 Recursion

- Example:
  - People are standing in queue
  - Doorman wants to know how many people are waiting
  - What are the base- and general cases?
```java
// Iterative way of computing the factorial of n
public static long factorialIterative(int n)
    long result = 1;
    for (int i = 1; i <= n; i++)
    {
        result = result * i;
    }
    return result;

// Recursive way of computing the factorial of n
public static long factorialRecursive(int n)
    if (n == 1)
    {
        return 1;
    }
    else
    {
        return n * factorialRecursive(n - 1);
    }
```
The recursive way of computing the factorial of 'n'.

```java
public static long factorialRecursive(int n) {
    if (n <= 1) {
        return 1;
    } else if (n == 1) {
        return 1;
    } else {
        long a = factorialRecursive(n-1);
        return n * a;
    }
}
```
4 Recursion

Recursive method calls & Stack

- Local variables and parameters stored on stack
- For each function call, a corresponding stack frame is created

```java
public static long factorialRecursive(int n) {
    if (n < 0) {
        System.err.println("Negative input!");
        return 1;
    } else if (n == 0) {
        return 1;
    } else {
        long a = factorialRecursive(n-1);
        return n * a;
    }
}
```

```java
public static long factorialIterative(int n) {
    long result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

```java
public static void main(String[] args) {
    long result = factorial(4);
    System.out.println(result);
}
```
```java
public static void printArray(int[] numbers) {
    for (int i = 0; i < numbers.length; i++) {
        System.out.print(numbers[i] + " ");
    }
}
```

```java
public static boolean binarySearch(int left, int[] numbers, int right) {
    int pivotIndex = (left + right) / 2;
    if (pivotIndex == left) {
        return true;
    } else if (left < numbers[pivotIndex]) {
        return true;
    } else if (right > numbers[pivotIndex]) {
        return true;
    }
    return false;
}
```
5 Error Handling & Exceptions

Exceptions

- What if something goes wrong? → Program termination??!! With every error?
  → Obviously not intelligent!
- Mechanism in Java: Exceptions
- Definition [ITutorial]: “An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program’s instructions.”
- Exceptions are like balls that are thrown when something unusual occurs. Somebody must catch the ball and handle the exception or the program must terminate.
5 Error Handling & Exceptions

Exceptions

- Exceptions are like balls that are thrown when something unusual (an error) occurs. Somebody must catch the ball and handle the exception or the program must terminate.

```
try {
    FileWriter fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write("a");
    // ...
} catch (IOException e) {
    // Exception handling code goes here
    e.printStackTrace();
} catch (SomeOtherException e) {
    // Exception handling code goes here
    e.printStackTrace();
} catch (Exception e) {
    // Exception handling code goes here
    e.printStackTrace();
}
```

- "finally" block is always executed, i.e. even if an exception occurs

```
FileWriter fileWriter = null;
try {
    fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write("a");
    // ...
} catch (Exception e) {
    // Exception handling code goes here
} finally {
    // Make sure that the file is closed, no matter whether
    // an exception occurred or not.
    if (fileWriter != null) {
        fileWriter.close();
    }
}
```
5 Error Handling & Exceptions

Exceptions

- Usual case: Methods "try" possibly dangerous code and "catch" (handle) correspondingly resulting exceptions themselves

```java
try {
    // ...
    FileWriter fileWriter = new FileWriter("someFileName.txt");
    fileWriter.write('a');
    // ...
}
catch (IOException e) {
    // Exception handling code goes here
    e.printStackTrace();
}
catch (SomeOtherException e) {
    // Exception handling code goes here
    e.printStackTrace();
}
catch (Exception e) {
    // Exception handling code goes here
    e.printStackTrace();
}
```

- Other possibility: Let others (callers) handle the problem!
  Add `throws` clause to method/constructor definition

```java
class InvalidGearException extends Exception {}
class TireExplodedException extends Exception {}

class Bicycle {
    int gear;

    Bicycle(int initialGear) throws InvalidGearException {
        if (initialGear > 0) {
            gear = initialGear;
        } else {
            throw new InvalidGearException();
        }
    }

    void inflateTires() throws TireExplodedException {
        // ...
    }
}
```

5 Error Handling & Exceptions

Exceptions

- Checked Exceptions
  - Subclasses of `java.lang.Exception`
  - Must be caught or forwarded where they possibly occur (try/catch or throws)
  - Example: Opening a file (is likely to go wrong)

- Unchecked ("Runtime") Exceptions
  - Subclasses of `java.lang.RuntimeException` or `java.lang.Error`
  - Can be caught, i.e. no try/catch or throws necessary
  - Example:

```java
static long factorial(int n) {
    if (n < 0) {
        throw new RuntimeException("Check your math!");
    } else if (n == 0) {
        return 1;
    }
    return n * factorial(n-1);
}
```

6 Coding & Naming Conventions

Deepening readings:

- [http://www.oracle.com/technetwork/java/codeconv-138413.html](http://www.oracle.com/technetwork/java/codeconv-138413.html)
- [http://geosoft.no/development/javastyle.html](http://geosoft.no/development/javastyle.html)
- [http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/index.html](http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/index.html)
6 Coding & Naming Conventions

Deepening readings:
http://www.oracle.com/technetwork/java/codestyle-138413.html
http://geosoft.no/development/javastyle.html
http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/index.html

Tip: Use IDE/editor with syntax highlighting!
7 Using the Java Class Library

Packages and Imports

- Java’s classes and types are organized in hierarchical packages
  - `java.lang.String`
  - `java.net.URLConnection`
  - `java.util.Collection`
  - `javax.xml.parsers.SAXParser`
  - `org.w3c.dom.events.DocumentEvent`

- All types from package `java.lang` are imported automatically
- Other types need to be imported
  - `import java.net.URLConnection;`
  - `import java.util.Collection;`
  - `import some.other.package.*;`  // * means all types

Generics

- Some types may be parameterized with other types
- Typical examples are classes that implement data structures
- Advantages:
  - Type checks at compile time
  - Programmers can implement algorithms generically

Examples:
```java
Vector nonGenericVector = new Vector();
nonGenericVector.add(1234);
nonGenericVector.add("Hello world");
Object typeUnknown = nonGenericVector.get(0);

Vector<Bicycle> bicycles = new Vector<Bicycle>();
bicycles.add(new Bicycle());
bicycles.add(123);  // Compile time error!
Bicycle typeKnown = bicycles.get(0);
```
7 Using the Java Class Library

Wrapper-Classes

- Parameters restricted to reference types (classes, interfaces)
- For each primitive type, a corresponding wrapper class exists
- Examples:
  
  java.lang.Short  
  java.lang.Integer  
  java.lang.Long  
  java.lang.Float  
  java.lang.Double  

  etc.

- Primitive types are automatically boxed and unboxed when necessary

  Integer i = 723;  
  int j = i;

8 Solving a Problem

Deepening readings (optional):

The Internet

Main reference:

http://docs.oracle.com/javase/6/docs/api/