# JAVA Programming

Overview of JAVA

# Overview

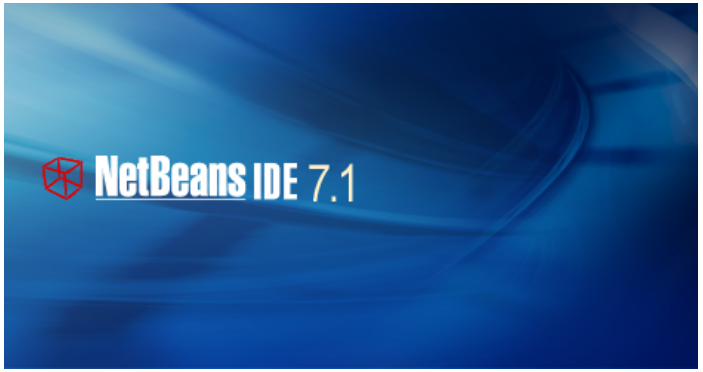
- Getting started with Java
- Class
- The main method
- Packages
- Import statements
- Naming
- Comments
- Basic Output operations
- Test
- Debug

# Getting Started with Java

- We don't want to program in notepad
  - However, it's possible
- We use an IDE
  - Integrated Development Environment
- Some IDE's
  - Eclipse
  - Netbeans
  - IntelliJ

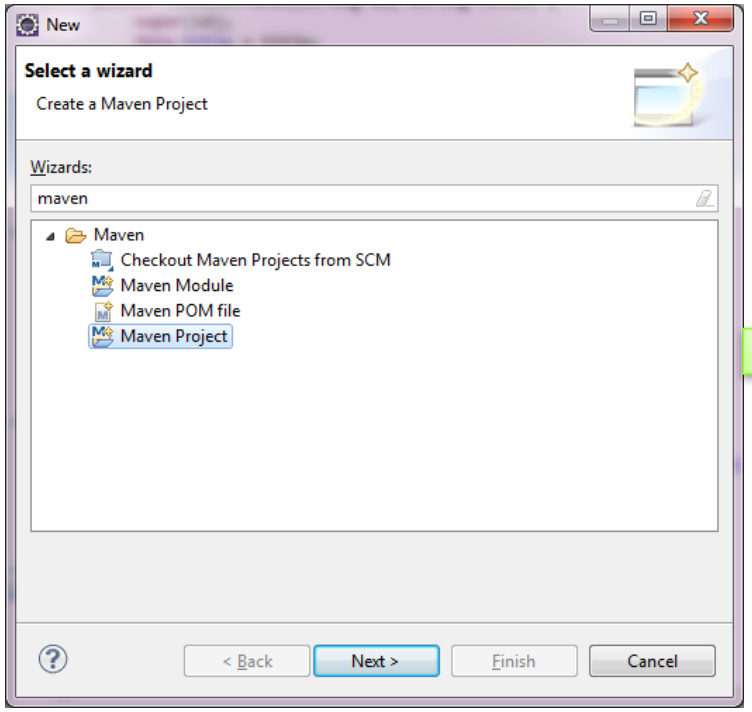
# Getting Started with Java

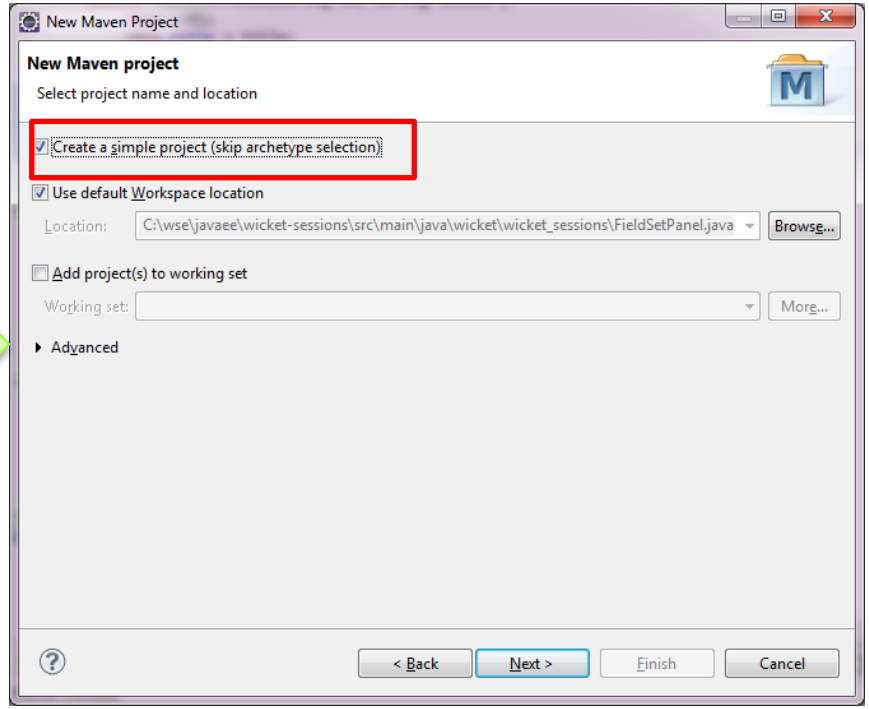
- Eclipse
  - Developed by IBM and donated to the Open Source community
  - Open source IDE
- Netbeans
  - Developed by Sun
  - Open source
- IntelliJ
  - Developed by JetBrains
  - Open source

# Getting Started with Java

- Start VMware image
- Start the IDE Eclipse
  - Create a project based on Maven (buildsystem)



# Getting Started with Java

- Provide project details

Unique name

Domain

New Maven Project

**New Maven project**

Configure project

Artifact

Group Id: nl.infosupport.kenniscentrum

Artifact Id: clientapp

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Parent Project

Group Id:

Artifact Id:

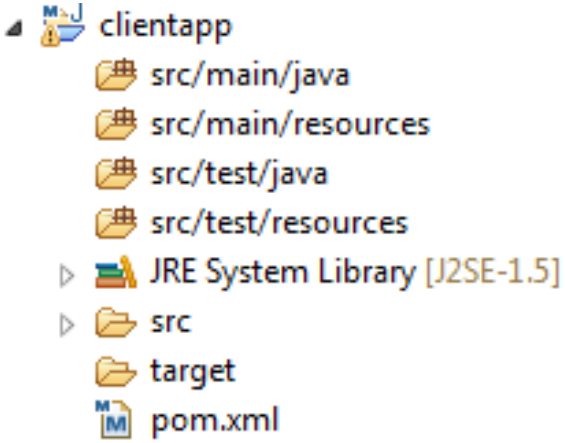
Version: Browse... Clear

Advanced

< Back Next > Finish Cancel

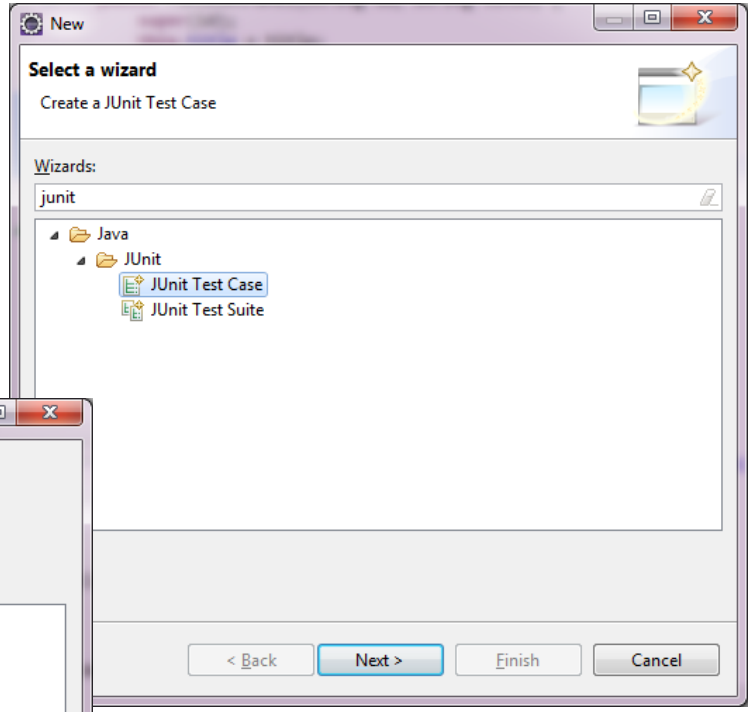
# Getting Started with Java

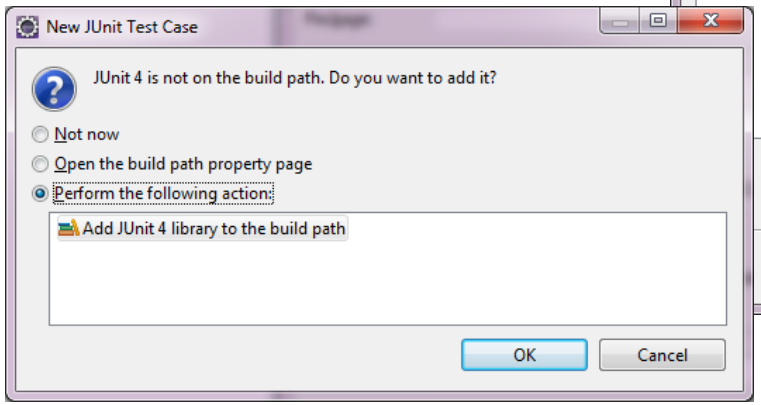
- Default Maven Project structure
  - Source code
  - Test code



# Getting Started with Java

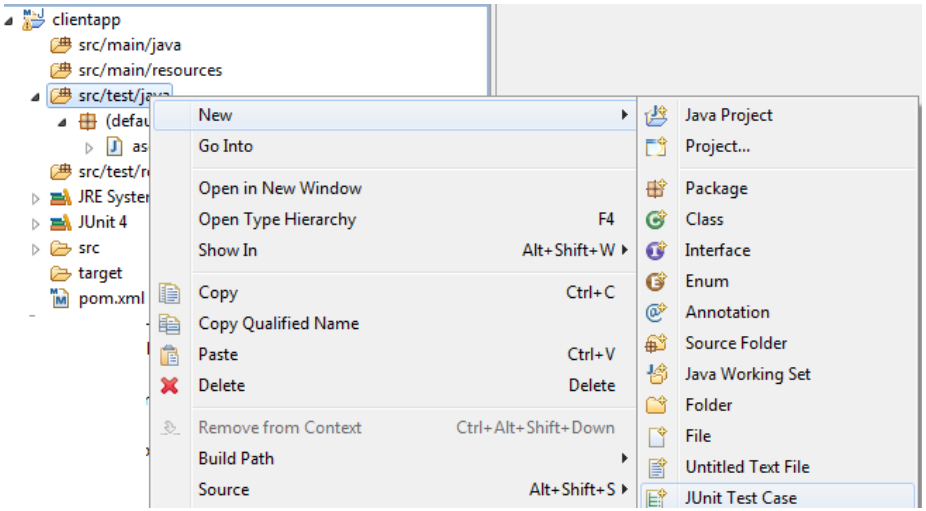
- Default Maven Project structure
  - Source code
  - Test code



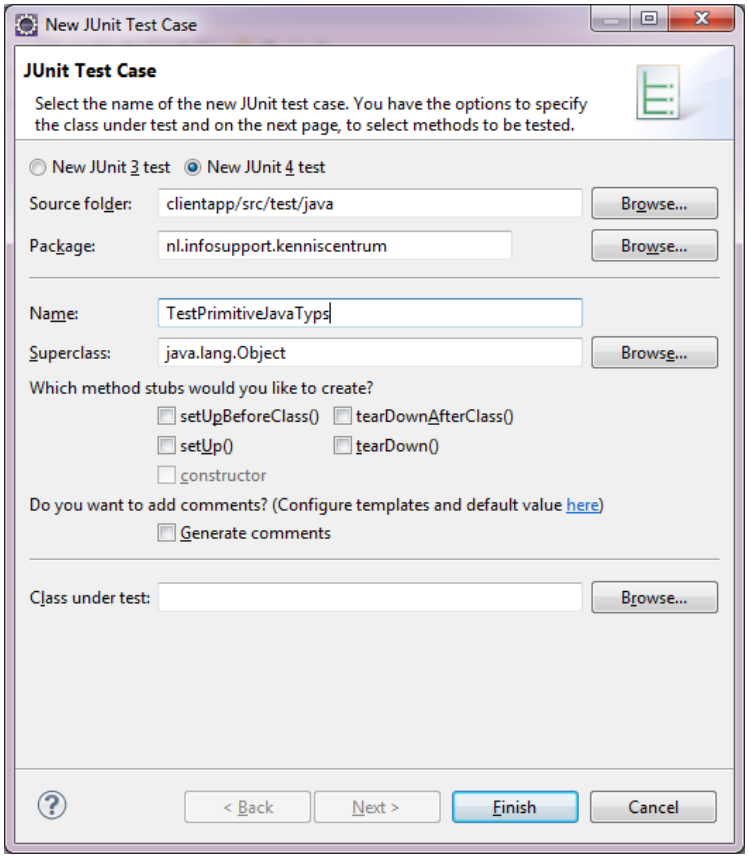
# Getting Started with Java

- Default Maven Project structure
  - Source code
  - Test code



# Getting Started with Java

- Default Maven Project structure
  - Source code
  - Test code



# Class

- A Java application consist of a collection of classes
- A class is a set of data and methods
- A class cannot span multiple files
- Only one class per file
- A Java program can consist of many files

# Class

Keywords shown in purple

Class name: `public class Client {…}`

File name: ….\src\Client.java

```
public class Client {
        public static void main(String[] args) {
                System.out.print("Enter the cardID:  ");
        }
}
```

code is case sensitive

String literal

# The main method

```
public class Client {
        public static void main(String[] args) {
                ….

}}
```

- Entry point is static main() method in a class
- Notice the signature

  ```
  public static void main(String[] args) {…}
  ```

- When main method ends, the application ends
- Command-line arguments appear as an array of strings (args)

# Packages

- Grouping of related classes, interfaces and resources
- Packages create namespaces to avoid name collisions
- Package name is specified in first line of the .java file
- Package name should be meaningful and unique e.g.

```
package com.company.math
```

- Use lower case

# Packages

- Can be distributed independently or with other packages, to form application
- Contains
  - Related classes
  - Interfaces
  - Sub packages
  - Additional resources
- Fully qualified name: packagename.classname
- Simple name: classname (import needed)

# Packages

```
package com.company.demo1;

  public class Person  {
        private String name;

        public String getName() {
          return name;
        }
}
```

Package name

```
package com.company.demo1;
 public class Employee extends Person {
        private int ID;
        public int getID() {
                return ID;
        }
}
```

Package name

# Import statements

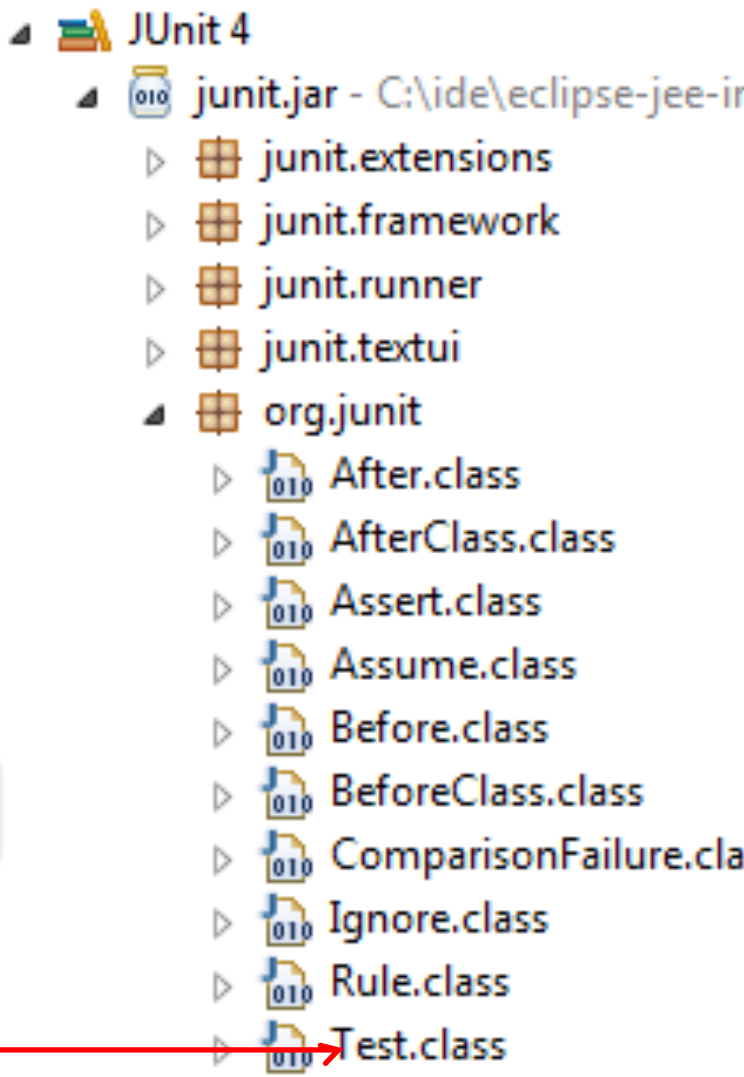
- Import statement

simple name

```
import org.junit.Test;
```

Fully qualified name

Use your IDE:
<CTRL> + <Shift> + O to Organize imports



# Static Imports

- In order to access static members, qualify references with the class they came from

```
double result= Math.cos(Math.PI * 1.23);
```

- The static import construct allows unqualified member access

```
import static java.lang.Math.*;
public class StaticImports{

public static void main(String[] args) {
//double result= Math.cos(Math.PI * 1.23);
double result= cos(PI * 1.23);
```

# Static Imports

- Use them sparingly.
  - Class name gives the member a descriptive namespace which clarifies its purpose.
- When you choose to use them: import all members explicitly so  you can see where each member origins from.

# Naming

- Use meaningful names for
  - Packages
  - Classes
  - Variables
  - Methods

# Comments

- Comments are important

  A well-commented application permits a developer to fully understand the structure of the application

- Single-line comments

```
// TODO method must be modified to improve performance
```

- Multiple-line comments

```
/*
* Method calculates the faculty
*/
```

- Documentation comment

```
/**
* Method calculates the faculty
*/
```

# Basic Output Operations

- Print a variable and go to new line

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

- Print a variable and stay on same line

```
System.out.print(path);
```

- Concatenate string literal and variable of type String

```
System.out.println("Path: "+path);
```

- Concatenate string literal and variable of type String using printf

```
System.out.printf("The file %s was not found", path);
```

# Test

- Create Test class with test methods to test your application

```
public class TestPrimitiveJavaTypes {

   @Test
  public void test() {
    System.out.println("Hello World");
  }
}
```

annotation

- Use descriptive names for your test methods

```
    public void messageIsPrintedToConsole()
```

# Test

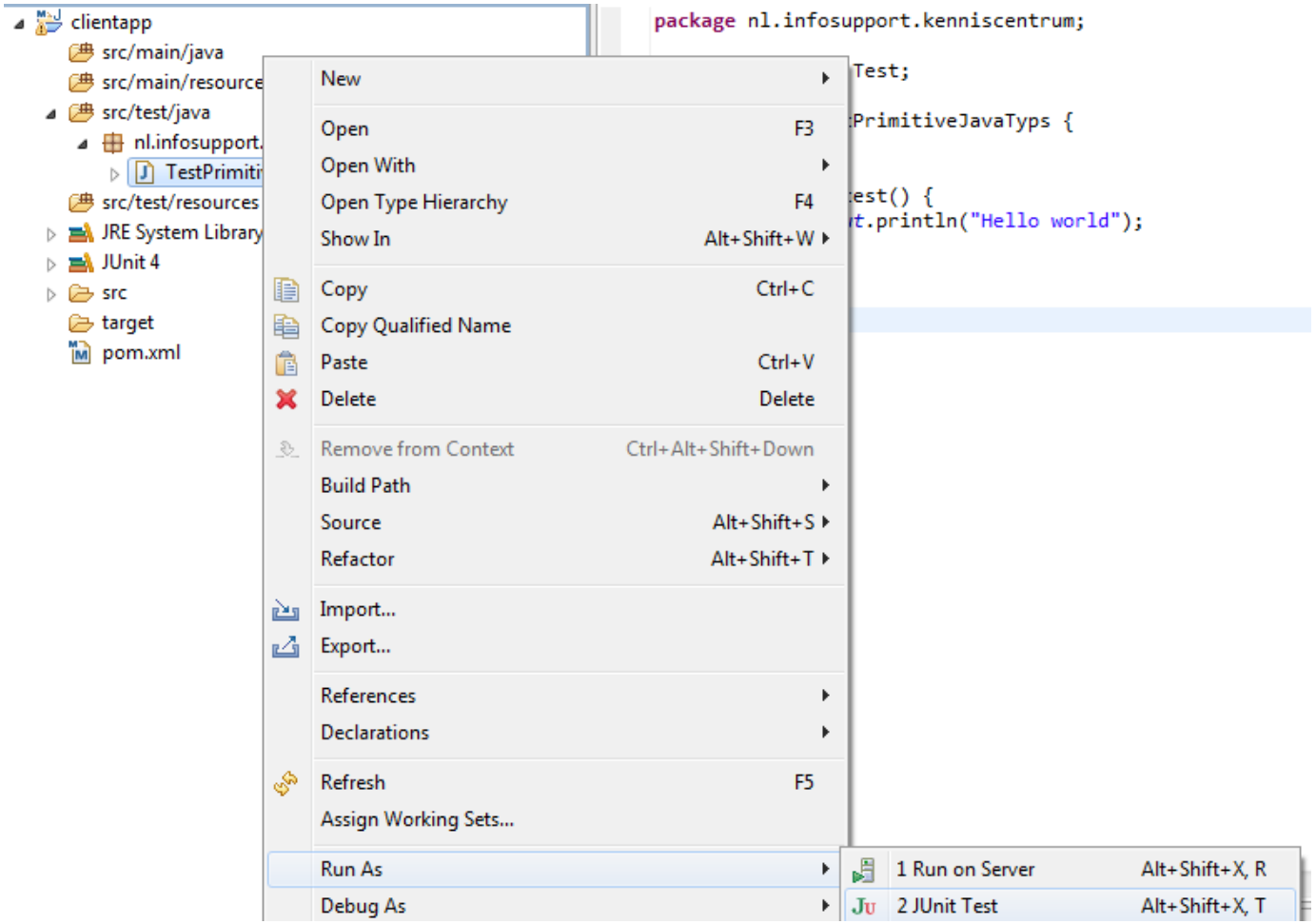
- Meta data for our test method
- @Test refers to org.junit.Test
- A reference is needed (import)

```
import org.junit.Test;
public class TestPrimitiveJavaTypes {

  @Test
  public void messageIsPrintedToConsole() {
    System.out.println("Hello World");
  }
}
```

# Run a test

- Run our first test



# Debugging

- Debugging
  - Setting breakpoints and watches
  - Stepping through code
  - Examining and modifying variables

# Lab 2

- Write an ‘Hello World’ application