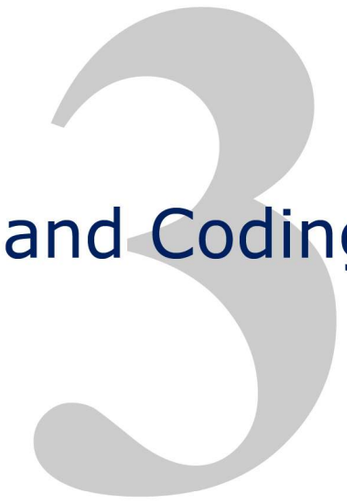


Basic Syntax and Coding Conventions

Objectives

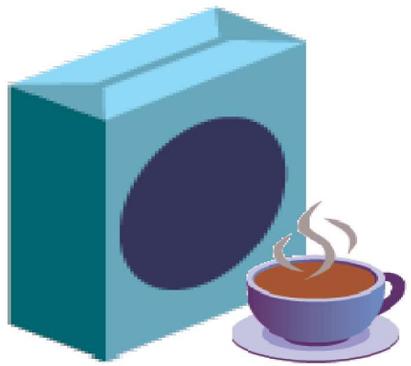
After completing this lesson, you should be able to do the following:

- Identify the key components of Java
- Identify the three top-level constructs in a Java program
- Identify and describe Java packages
- Describe the basic language syntax and identify Java keywords
- Identify the basic constructs of a Java program
- Compile and run a Java application
- Use the `CLASSPATH` variable and understand its importance during compile and run time

Toolkit Components

Java SE and Java EE provide:

- Compiler
- Bytecode interpreter
- Documentation generator

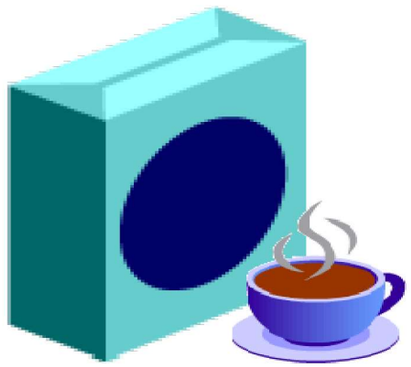


Java SE

Java Packages

Java SE and Java EE provide standard packages for:

- Language
- Windowing
- Input/output
- Network communication
- Other packages

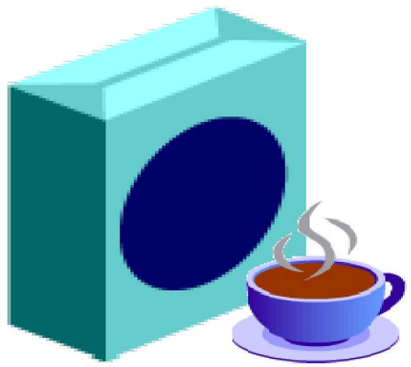


Java SE

Documenting Using Java SE

Java SE and Java EE provide documentation support for:

- Comments
  - Implementation
  - Documentation
- Documentation generator



Java SE

Annotations are a new feature in Java SE 5.0:

- Generate boilerplate code from annotations in source code
- Eliminate need for "side files" to be kept up-to-date with changes in the source
- Complement `javadoc` tags

Contents of a Java Source File

- A Java source file can contain three top-level constructs:
  - Only one `package` keyword followed by the package name, per file (always the first line in the file)
  - Zero or more `import` statements followed by fully qualified class names or "*" qualified by a package name
  - One or more `class` or `interface` definitions followed by a name and block
- File name must have the same name as the public class or public interface.

Naming Conventions

Naming conventions include:

- Class names
  - `Customer`, `RentalItem`, `InventoryItem`
- File names
  - `Customer.java`, `RentalItem.java`
- Method names
  - `getCustomerName()`, `setRentalItemPrice()`
- Package names
  - `oracle.xml.xsql`, `java.awt`, `java.io`

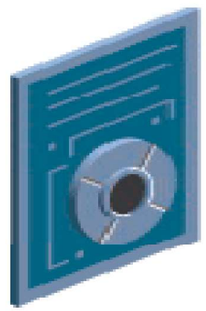
More About Naming Conventions

- Variables:
  - `customerName`, `customerCreditLimit`
- Constants:
  - `MIN_WIDTH`, `MAX_NUMBER_OF_ITEMS`
- Uppercase and lowercase characters
- Numerics and special characters

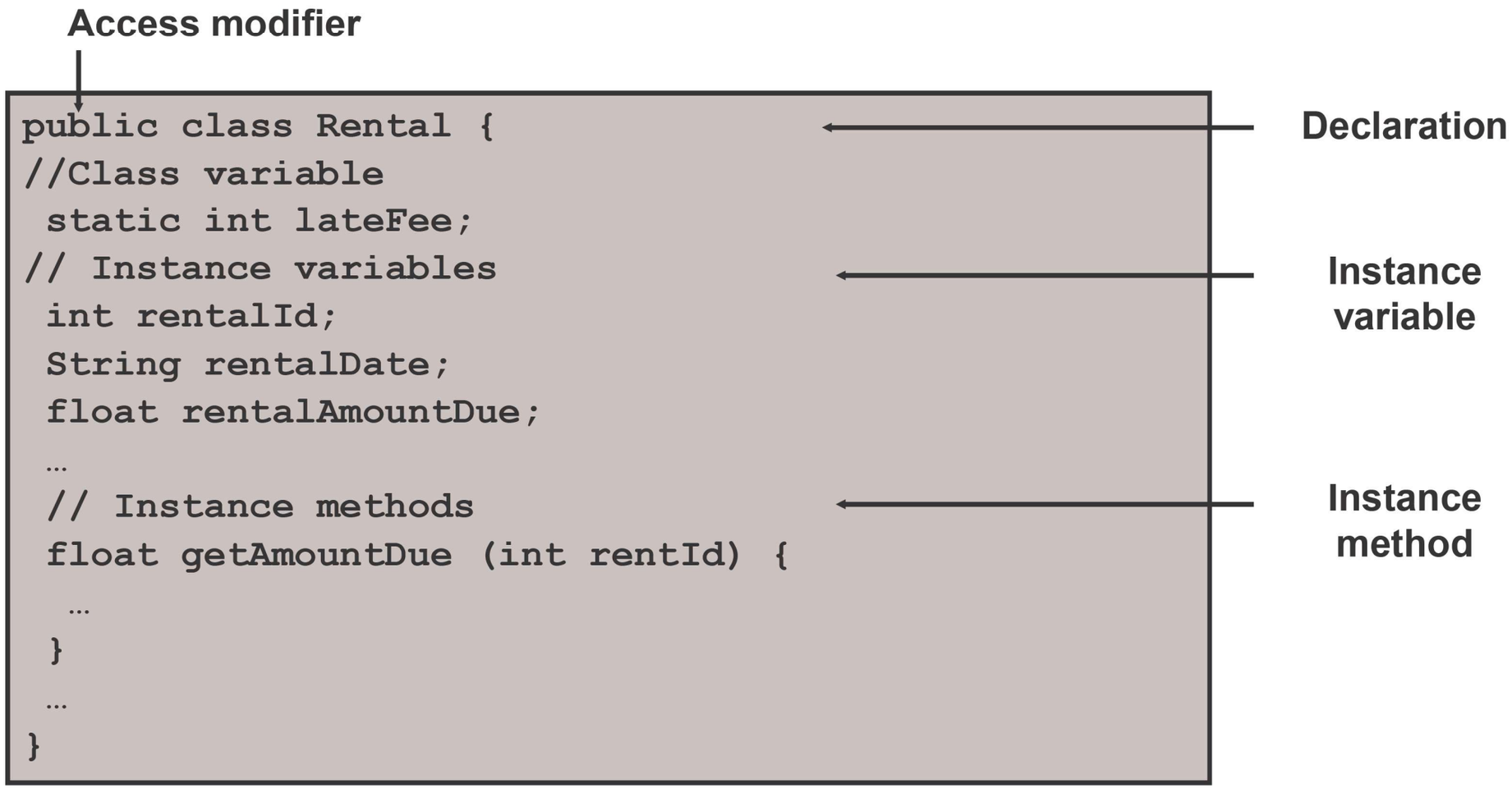
Defining a Class

Class definitions typically include:

- Access modifier
- Class keyword
- Instance fields
- Constructors
- Instance methods
- Class fields
- Class methods

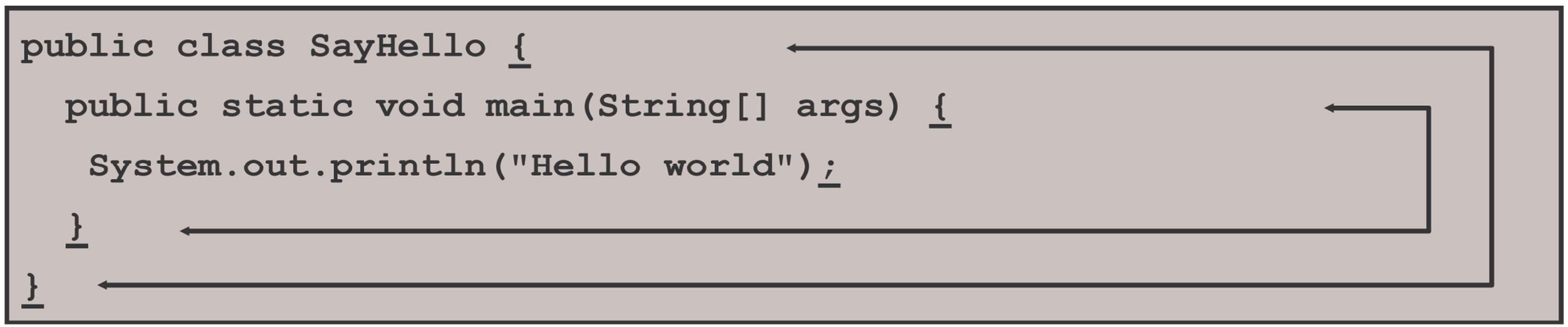


`Rental` Class: Example



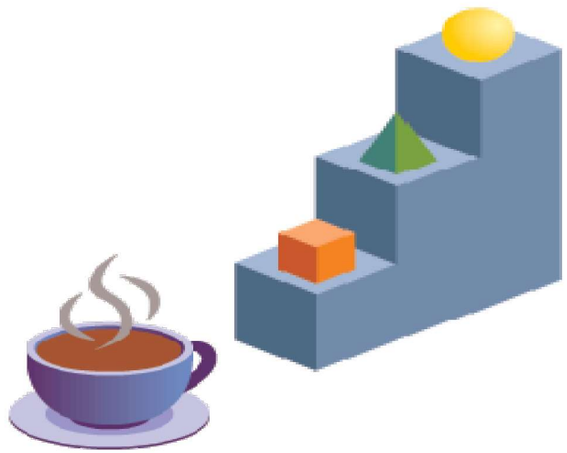
Creating Code Blocks

- Enclose all class declarations.
- Enclose all method declarations.
- Group other related code segments.



Defining Java Methods

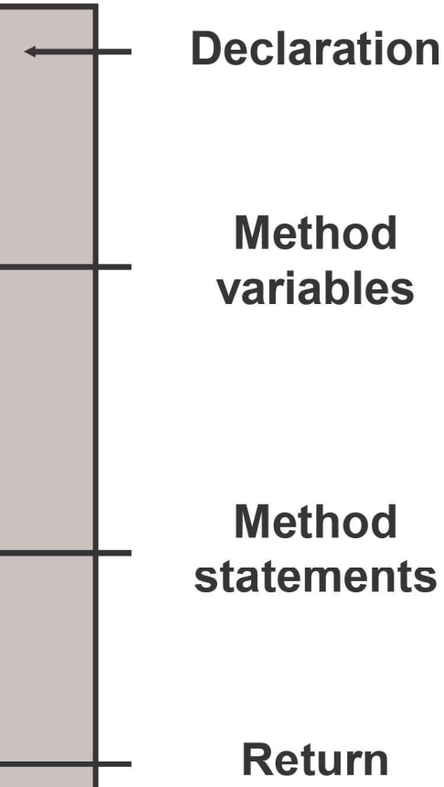
- Always define within a class.
- Specify:
  - Access modifier
  - Static keyword
  - Arguments



```
[access-modifiers] [static] "return-type"
  "method-name" ([arguments]) {
      "java code block" …
      return;
 }
```

Example of a Method

```
public float getAmountDue (String cust){
 // method variables
 int numberOfDays;
 float due;
 float lateFee = 1.50F;
 String customerName;
// method body
 numberOfDays = getOverDueDays();
 due = numberOfDays * lateFee;
 customerName = getCustomerName(cust);
. . .
 return due;
}
```



Declaring Variables

- You can declare variables anywhere in a class block and outside any method.
- You must declare variables before they are used inside a method.
- It is typical to declare variables at the beginning of a class block.
- The scope or visibility of variables is determined in the code block.
- You must initialize method variables before using them.
- Class and instance variables are automatically initialized.

Examples of Variables in the Context of a Method

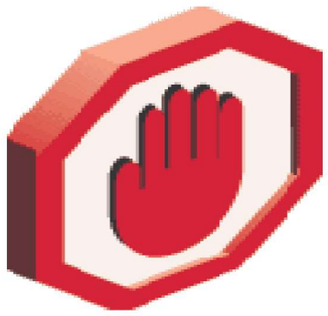
```
public float getAmountDue (String cust) {
 float due = 0;
 int numberOfDays = 0;
 float lateFee = 1.50F;
 {int tempCount = 1; // new code block
   due = numberOfDays * lateFee;
   tempCount++;
   …
 }                    // end code block
 return due;
}
```

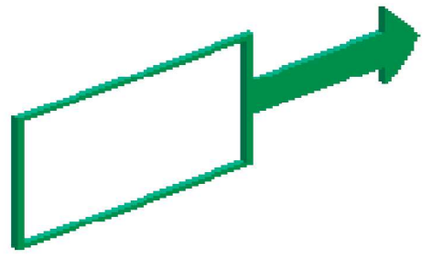
Method variables

Temporary variables

Rules for Creating Statements

- Use a semicolon to terminate statements.
- Define multiple statements within braces.
- Use braces for control statements.





Compiling and Running

- To compile a `.java` file:

```
prompt> javac SayHello.java
… compiler output …
```

- To execute a `.class` file:

```
prompt> java SayHello
Hello world
prompt>
```

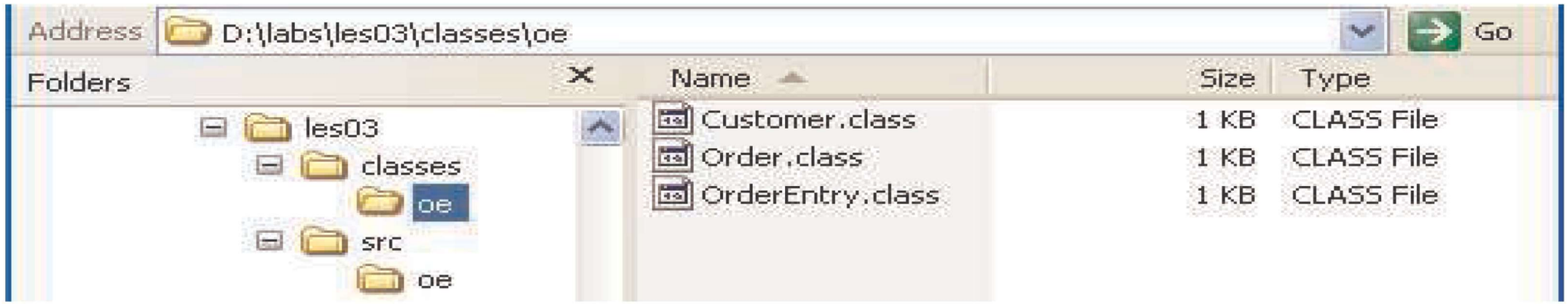
- Remember that case matters.

CLASSPATH Variable

- Is defined in the operating system
- Directs the JVM and Java applications where to find `.class` files
- References built-in libraries or user-defined libraries
- Enables the interpreter to search paths; loads built-in classes before user-defined classes
- Can be used with `javac` and `java` commands

Classpath Use Examples

Location of `.class` files in the `oe` package



Setting `CLASSPATH` at OS level

```
C:\>set CLASSPATH=D:labs\les03\classes
```

Using the `-classpath` option

```
C:\> javac -classpath D:\labs\les03\classes -d D:\labs\les03\classes
         D:\labs\les03\src\oe\OrderEntry.java
C:\> java -classpath D:\labs\les03\classes oe.OrderEntry
```

Summary

In this lesson, you should have learned the following:

- Java SE provides basic Java tools.
- Java SE provides a rich set of predefined classes and methods.
- Java programs are made up of classes, objects, and methods.

Practice : Overview

The two practices for this lesson cover the following topics:

- Examining the Java environment
- Writing and running a simple Java application
- Examining the course solution application
- Inspecting classes, methods, and variables
- Creating class files and an application class with a `main( )` method
- Compiling and running an application