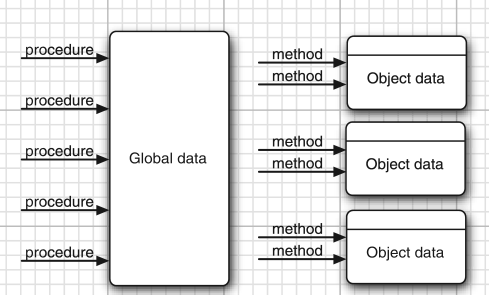
Chapter 4: Objects and Classes

# 4.1. Introdu ction to Object-Oriented Programming

-An object-oriented is made of objects which will be taken from a library or custom-designed.

OOP order: put the data first, then look at algorithms to operate on data



## 4.1.1 Classes

-A **class** specifies how **objects** are made. When you **construct** an object from a class, you have created an **instance** of the class.

-All code you write is inside a class.

-Standard Java library supplies thousand classes: UI design, dates and calendars, network

-**Encapsulation**(**information** **hiding**): combine data and behavior in one package and hide the implementation details from users of object. The bits of data in object are **instance fields**, procedures are **methods***.* An object is an instance of class and have values of instance fields: called **state**. The key to making **encapsulation** is to have methods never directly access instance in class other than their own.

-Classes can be build by **extending** other classes. Java comes with superclass **Object**, all other classes extend this class. When extend a class, new class has all properties and methods of class. The concept of extending to make new class is **inheritance**

## 4.1.2 Objects

-3 key characteristics of objects: behavior, state, indentity

-**Behavior**: defined by methods

-**State**: what object looks like

-**Identity**: the objects that are instances of a class always differ in their identify.

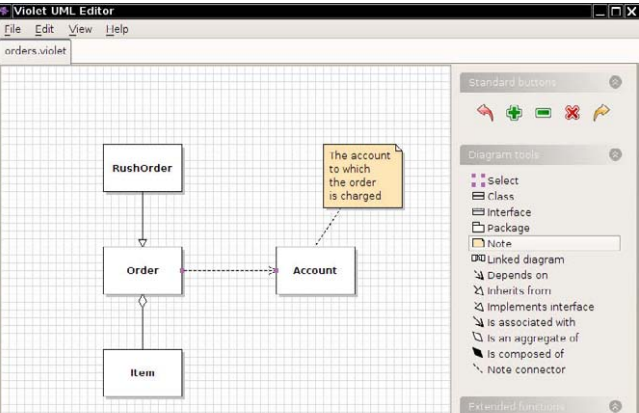
## 4.1.3 Identifying Classes

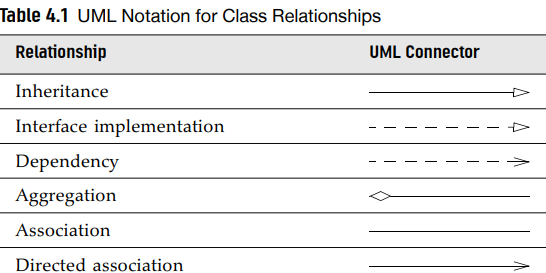
-First work is to identify class and add methods to each class: look for nouns in problem analysis. Methods correspond to verbs.

## 4.1.4 Relationships between Classes

-Most common relationships classes: **Dependence**(uses-a), **Aggregation**(has-a), **inheritance**(is-a)

-**UML**: draw **k** describe relationships between classes.



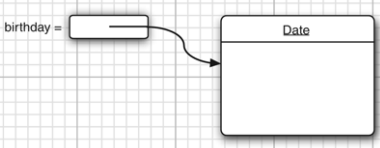


# 4.2 Using Predefined Classes

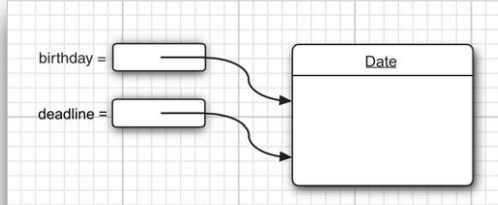
## 4.2.1 Objects and Object Variables

-First, construct object:









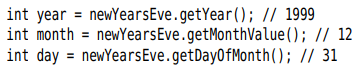
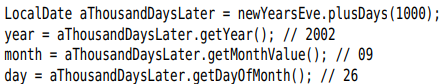
-Object variable only **refers** to an object. The value of any object variable is a reference to an object that is stored elsewhere.

## 4.2.2 The LocalDate Class of the Java Library

-Java library contains 2 classes: Date class (represent a point in time) and LocalDate (Expresses days in calendar)





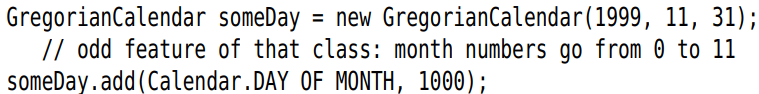
-Note: Date class also has methods to get day, month, year but these are deprecated.

## 4.2.3 Mutator and Accessor Methods



+pluDays method yields a new LocalDate object then assigned to aThousandDaysLater variable, the originally object remains unchanged => plusDays method does not **mutate** the object.

-The GregorianCalendar.add method is a **mutator method***:*

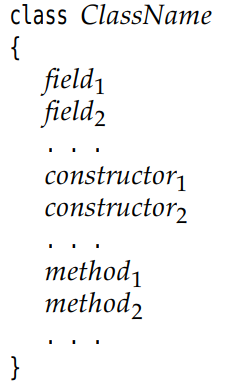


-Method that only access objects without modifying them are **accessor methods**: LocalDate.getYear, GregorianCalendar.get

-API note:p141

# 4.3 Defining your own Classes

## 4.3.1 An Employee Class

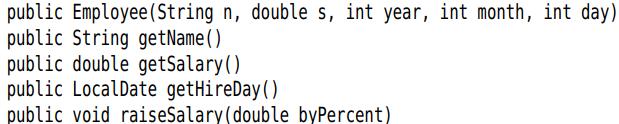


## 4.3.2 Use of Multiple Source Files

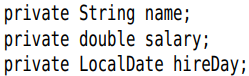
-Just compile the class including main method. Class inside will be automatically searched and compiled (or recompiled if having a new version)

## 4.3.3 Dissecting the Employee Class

-1 constructor and 4 methods



-3 instance fields



## 4.3.4 First Steps with Constructors

-A constructor can only be called in conjunction with **new** operator:



## 4.3.5 Declaring Local Variables with var



## 4.3.6 Working with null References

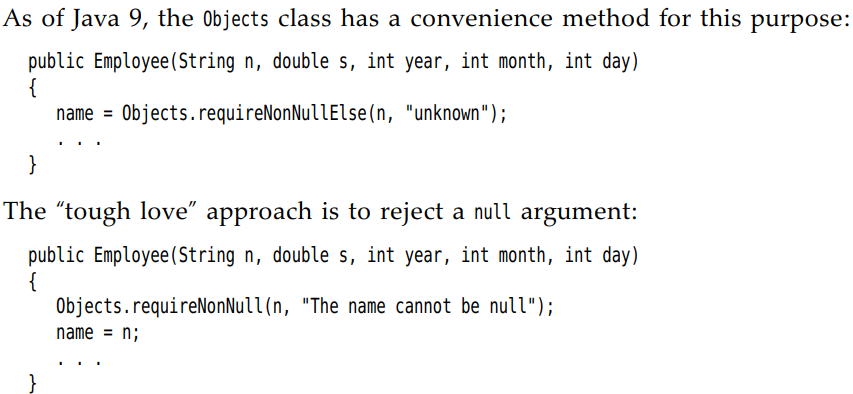
-**null** indicates the absence of an object in variable.

-If you apply a method to a null value => **NullPointerException**



-Solution:





## 4.3.7 Implicit and Explicit Parameters

-Implicit parameter: the object of type, the keyword *this*

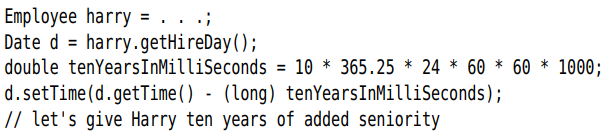
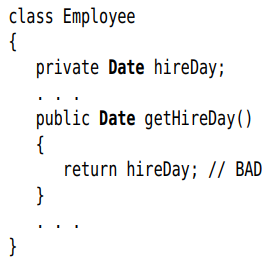
-Explicit parameter: other parameters

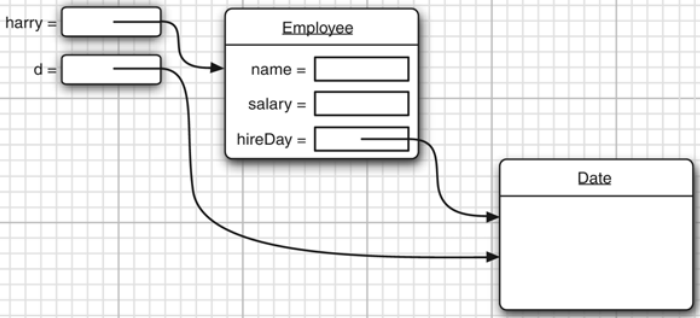
## 4.3.8 Benefits of Encapsulation

-**field accessors***:* accessor methods, return the values of instance fields.

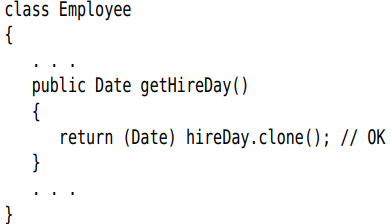
-Get and set the value of an instance field: private data field, public field accessor method and public field mutator method.

-Note: Not to write accessor methods return references to mutable objects:



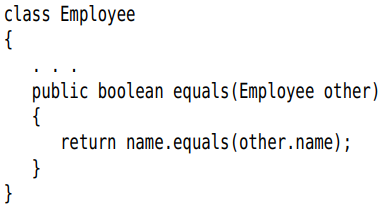


+You should clone it first:



## 4.3.9 Class-Based Access Privileges

-A method can access all objects of its class:

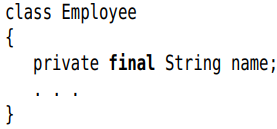


## 4.3.10 Private Methods

-We make all data fields private. Most methods are public. Sometimes methods are best implemented as private.

## 4.3.11 Final Instance Fields

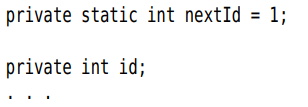
-After setting value of constructor, final fields may not be modified again



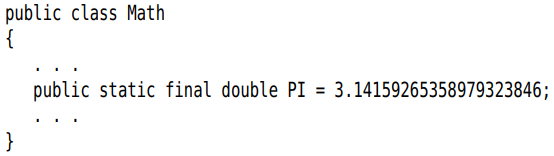
# 4.4 Static Fields and Methods

## 4.4.1 Static Fields

-Static field: there is only one field per class. It belongs to the class, not to any individual object.

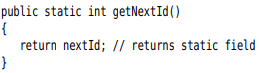


## 4.4.2 Static Constants



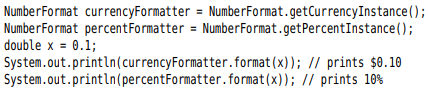
## 4.4.3 Static Methods

-Methods that do not operate on objects that don’t have a this parameter.



## 4.4.4 Factory Methods

-Factory methods that construct objects: LocalDate.now, LocalDate.of



## 4.4.5 The main Method

-When a program starts, there aren’t any objects yet, the static main method executes and constructs the objects that program needs.

-Note: Every class can have a main method for unit testing of classes.

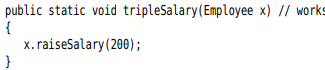
# 4.5 Method Parameters

-*Call by value:* the methods gets just the value

-*Call by reference*: the method gets the location of the variable.

-Java always uses call by value

-There are 2 kinds of method parameters: primitive types(number, boolean) and object references. Method can change object parameters.



-Summary:

+Method cannot modify a parameter of a primitive type

+Method can change the **state** of an object parameter

+Method cannot make an object parameter refer to a new object

# 4.6 Object Construction

4.6.1 Overloading

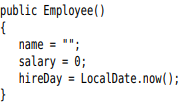
-**Overloading**: several methods have the same name but different parameters and different return types.

## 4.6.2 Default Field Initialization

-If you don’t set a field explicitly in a constructor, it is automatically set to a **default value**: numbers to 0, boolean to false, object reference to null.

## 4.6.3 The Constructor with No Arguments

-Constructor with no arguments: creates an object whose state is set to an appropriate default.

-If a class with no constructors: a no-argument constructor is provided with all default values fields.

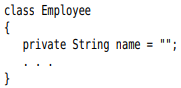
-If a class supplies at least 1 constructor but does not supply no-argument constructor => cannot construct object with no-arguments



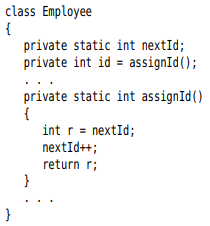
 would be an error

## 4.6.4 Explicit Field Initialization

-You can simply assign a value to any field in class definition. This assignment is carried out before the constructor executes.

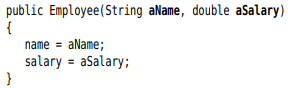


-The initialization value doesn’t have to be a constant value:

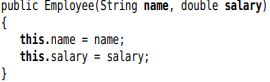


## 4.6.5 Parameter Names

-Prefix each parameter with “a”

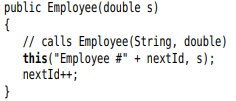


-this:



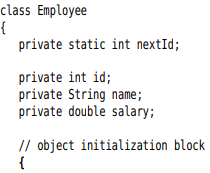
## 4.6.6 Calling Another Constructor

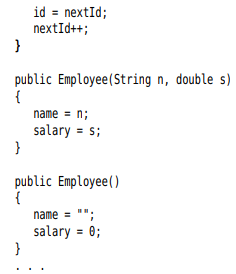
-If the first statement of a constructor has form *this(…)*, the constructor calls another constructor.

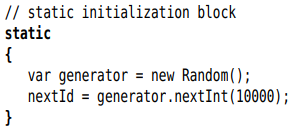


## 4.6.7 Initialization Blocks

-**Initialization block**:







-Summary: Initialize a data field

+In constructor

+In declaration

+Initialization block

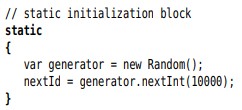
-Summary: Constructor flow

+First line calls a second constructor? Executes

+fields: default values + fields initializers and initialization block in the order in class declaration

+The body of constructor.

-Static initialization block: complex static field



-API note: p180

## 4.6.8 Object Destruction and the finalize method

-Java does **automatic garbage collection**, manual memory reclamation is not needed.

-Some objects utilize a resource other than memory (file) =>reclaimed and recycled when no needed.

-You can use **close** method. If you can wait until the virtual machine exits, use **Runtime.addShutdownHook**. Java 9: you can use **Cleaner** class that carried out when object is no reachable.

# 4.7 Packages

-You can group classes in a collection called *package*. It helps you to organize and separate the work from code libraries by others.

## 4.7.1 Package Names

-Use packages to guarantee the uniqueness of class names. To guarantee a unique package name, use Internet domain name written in reverse, then use subpackages for different projects.

-Note: No relationship between nested packages from the point of view of the compiler

## 4.7.2 Class importation

-A class can use all classes from its own package and all *public* classes from other packages.

-2 ways to access public classes in another package:

+Use *fully qualified name*(package name+ class name):



+Use import statement:

package

or specific class: 

-Use \* notation to import a single package.

## 4.7.3 Static imports

-Use the static methods and fields of class:



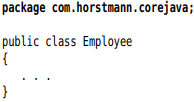


-Import a specific method or field:



## 4.7.4 Addition of a class into a Package

-To place classes inside a package, put the name of package at the top of source file, before the code.



-If you don’t put a *package* statement, file belongs to *unnamed package*

-Place source files into a subdirectory that matches the full package name

-Note: the compiler: javac, the interpreter: java

## 4.7.5 Package Access

-If you don’t specify the feature (class, method, variable) either public or private, it can be accessed by all methods in the same package.

## 4.7.6 The Class Path

-Classes are stored in subdirectories of file system. The path to the class must match the package name.

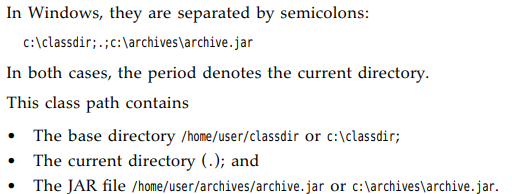
-Class files can also be stored in JAR (Java archive) file. It contains class files and subdirectories in compressed format => saving space and improving performance.

-To share classes among programs:

+Place the class files inside a directory – base directory

+Place any JAR files inside a directory

+Set the class path: the collection of all location that contain class files.



-You can specify a wildcard for JAR file directory:



-The class path lists all directories and archive files that are *starting points* for locating classes.

## 4.7.7 Setting the class path

-Specify the class path with the option -*classpath*



-CLASSPATH environment variable. The class path is set until the shell exits:



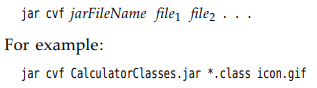
# 4.8 JAR Files

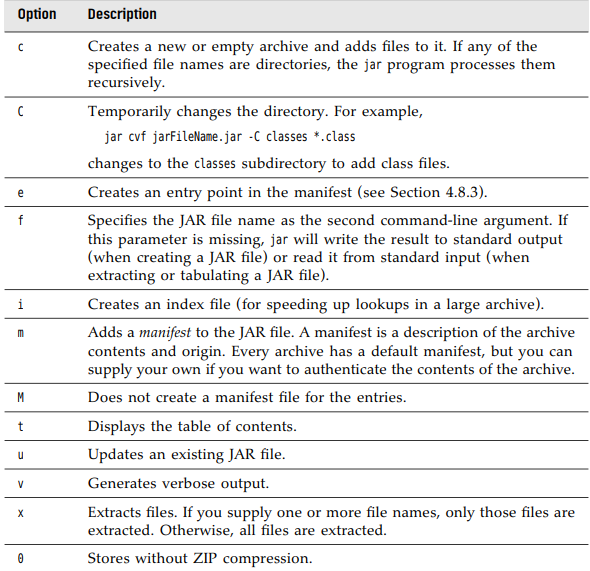
-When you package your application, you want to give uses a single file =>JAR file compressed by ZIP.

4.8.1 Creating JAR files

-**JAR file** contain class files and other file types, give users a single file.

-Use *jar* tool:





4.8.2 The Manifest

<https://docs.oracle.com/javase/10/docs/specs/jar/jar.html>

-manifest file describes special features of the archive, called **MAINFEST.MF** and located in **META-INF** bd

4.8.3 Executable JAR files

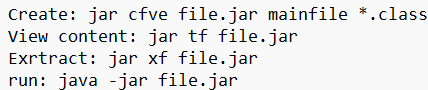
-Use the e option of jar command to specify the entry point of the program:



-Specify the main class in manifest:

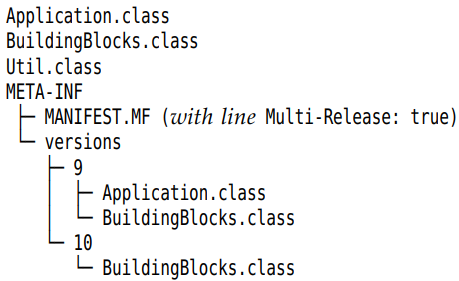


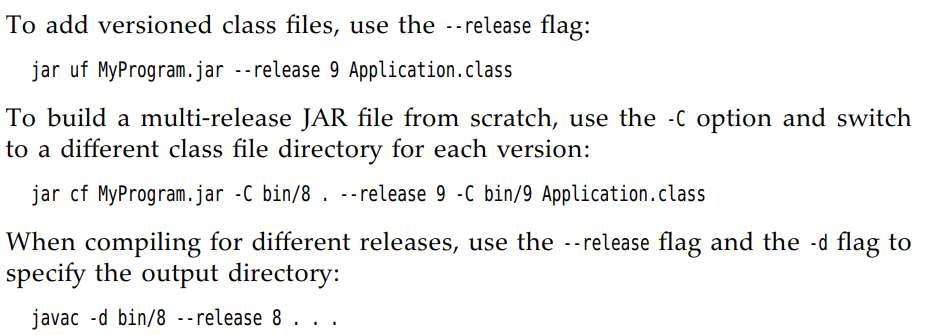
-Either method, users can start the program:



4.8.4 Multi-Release JAR files

-**multi-release JARs** contain class files for different Java releases.









Or



-Remember to run javac and jar in **Base directory**.

4.8.5 A note about Command-Line options

-**javadoc** tool generates HTML documentation form source files.

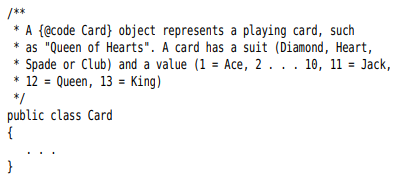
# 4.9 Documentation Comments

*Javadoc* tool generates HTML documentation form source files.

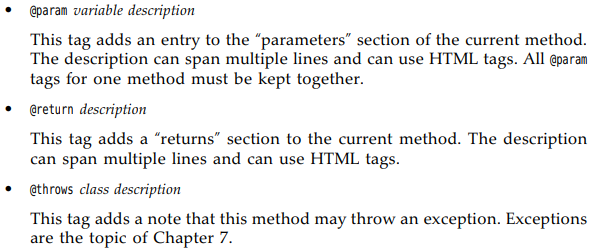
4.9.1 Comment Insertion

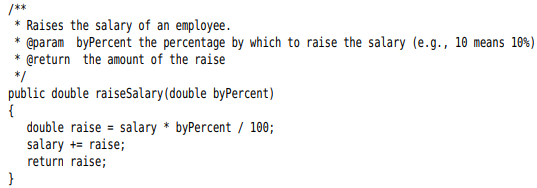
-Each comment is placed above the feature, starts with /\*\* and end with \*/. It contains *first sentence (summary statement)* and *free-form text* followed by *tags @.*

4.9.2 Class Comments

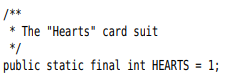


4.9.3 Method comments





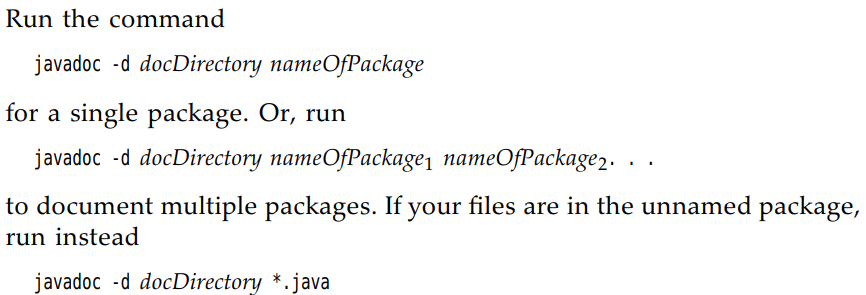
4.9.4 Field comments



4.9.5 General comments

4.9.6 Package comments

4.9.7 Comment Extraction



-**Read later**

# 4.10 Class Design Hints

-Always keep data private

-Always initialize data.

-Don’t use too many basic types in a class

-Not all fields need individual field accessors and mutators

-Break up classes that have too many responsibilities

-Make the names of your classes and methods reflect their responsibilities

-Prefer immutable classes.