Java basics

Estructura de clase en java y archivo de código fuente

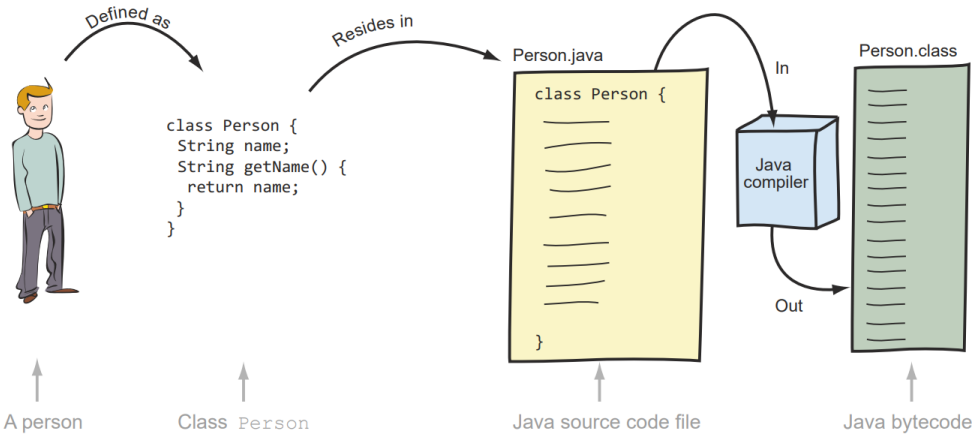


Figure 1.1 Relationship between the class file `Person` and the files Person.java and Person.class and how one transforms into another

Una persona puede definirse como la clase persona, esta clase reside en el archivo de código persona.java. Usando este código, el compilador de java (javac.exe) genera el bytecode y se almacena en el archivo persona.class.

Estructura de una clase de java

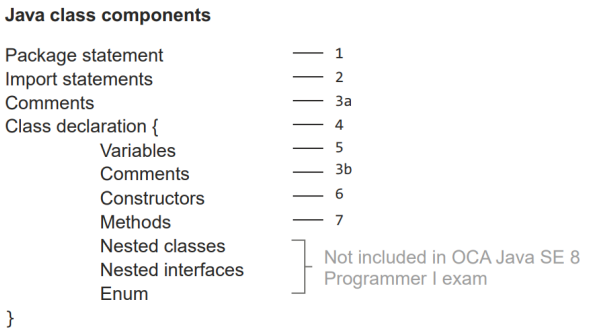


Figure 1.2 Components of a Java class

Package statement

- Toda clase es parte de un paquete
- La clase no puede pertenecer a dos paquetes
- La definición del paquete debe ser la primera línea, de otro modo existirá un fallo al compilar.
- La clase puede estar definida en un paquete nombrado explicitamente, de otra forma se asigna por defecto al paquete default, el cual no tiene nombre.

- La sentencia package se utiliza para saber en cual paquete se encuentra la clase.

Import statement

- Clases e interfaces en el mismo paquete pueden usarse entre ellas sin usar su prefijo de paquete.
- Para usar una clase o interfaz en otro paquete hay que usar su nombre completo: `packageName.anySubpackageName.ClassName`
- Por ejemplo, el nombre calificado de la clase `String` es `java.lang.String`

Comentarios

- Existen dos tipos de comentarios: multilínea y end-of-line
  - Multilínea
    - Empieza con `/*` y termina con `*/`
    - Ocupa multiples líneas de código
    - Pueden contener caracteres especiales
    - Por cuestiones estéticas es posible añadir un `*` al inicio de cada línea y controlar el formato
  - Final de línea
    - Empieza con `//`
    - Ocupa multiples líneas de código
    - Pueden contener caracteres especiales
    - Por cuestiones estéticas es posible añadir un `*` al inicio de cada línea y controlar el formato
- Los javadoc comments son comentarios especiales que inician con `/**` y terminan con `*/` . Son procesados para generar documentación.

Declaración de clase

La declaración de clase se realiza de la siguiente forma:

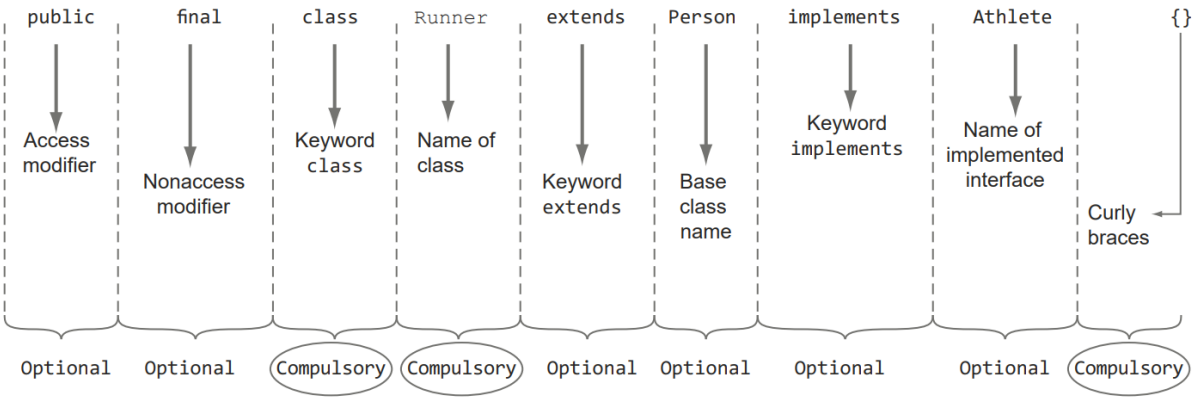


Figure 1.5 Components of a class declaration

Una clase es un diseño utilizado para especificar atributos y comportamientos de un objeto. Los atributos de un objeto son implementados usando variables y el comportamiento se implementa usando métodos.

- Los atributos pueden estar definiidos incluso después de los métodos

Variables

- Las variables son utilizadas para almacenar el estado de una instancia (variables de instancia o atributos de instancia)
- Las variables de instancia están definidas dentro de la clase pero fuera de sus métodos
- Una variable de clase o `static variable` es compartida por todos los objetos de esa clase.

Métodos

- Los métodos de instancia son utilizados usualmente para manipular variables de instancia
- Un método de clase o `static method` puede ser usado para manipular variables estáticas

Constructores

- Es utilizado para crear e inicializar objetos de una clase.
- Una clase puede definir multiples constructores que acepten diferentes parámetros

Estructura y componentes de un archivo de código

Un archivo de código es usado para definir entidades como clases, interfaces, enum. Estos archivos finalizan con la extención .java

Definición de una interface

Una interfaz puede ser definida como un contrato que las clases que la implementen deben cumplir.

- Sus métodos implicitamente son `abstract` y pueden contener tambien métodos concretos
- Sus métodos pueden definirse como `static`
- Se define con la palabra `interface`

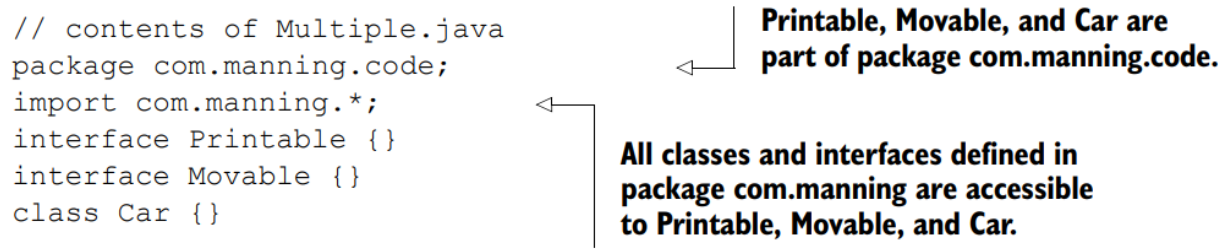
Definición de multiples clases en un sólo archivo

```
interface Printable {
    //.. we are not detailing this part
}
class MyClass {
    //.. we are not detailing this part
}
interface Movable {
    //.. we are not detailing this part
}
class Car {
    //.. we are not detailing this part
}
```

Contents of Java source code file Multiple2.java

- Es posible definir multiples entidades en un sólo archivo
- No hay un orden en particular para definir las entidades
- Un archivo no puede contener más de una clase pública o interfaz pública
- El nombre del archivo debe coincidir con el nombre de la clase pública o interfaz

Aplicación de package e import



- Tanto `package` como `import` aplican para todas las clases e interfaces definidas en el archivo

Aplicaciones ejecutables

Clases ejecutables vs no ejecutables

- La JVM comienza ejecutando el código definido en el método main
- Una clase no ejecutable es aquella que no contiene el método main

El método main

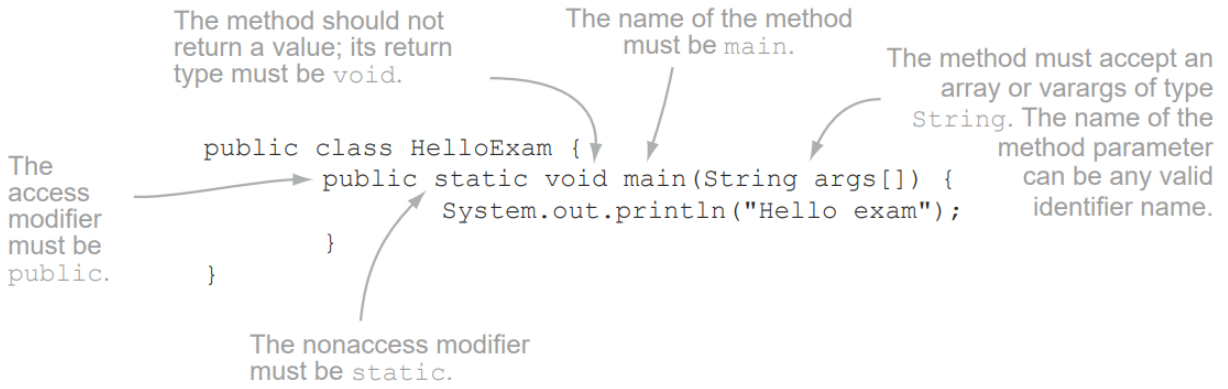


Figure 1.7 Ingredients of a correct `main` method

- Debe ser `public`
- Debe ser `static`
- El nombre del método debe ser main
- El tipo de retorno debe ser `void`
- Debe aceptar un argumento de tipo arreglo de `String` con cualquier nombre
- Es posible usar la ellipsis (...) siguiendo al tipo de variable → `public static void main(String… args)`
- El tipo de retorno siempre va acompañado del nombre del método
- El método main puede estar sobrecargado, sin embargo, en caso de no cumplir con la firma la clase será no ejecutable

Ejecutando desde la linea de comandos

- Para ejecutar una programa se emple a `java MyProgram arg1 arg2 argN`
- Los parámetros que son pasados son llamados parámetros de línea de comando o valores de línea de comando

Paquetes de Java

Table 1.2 Package-naming conventions used in the package name `com.oracle.javacert.associate`

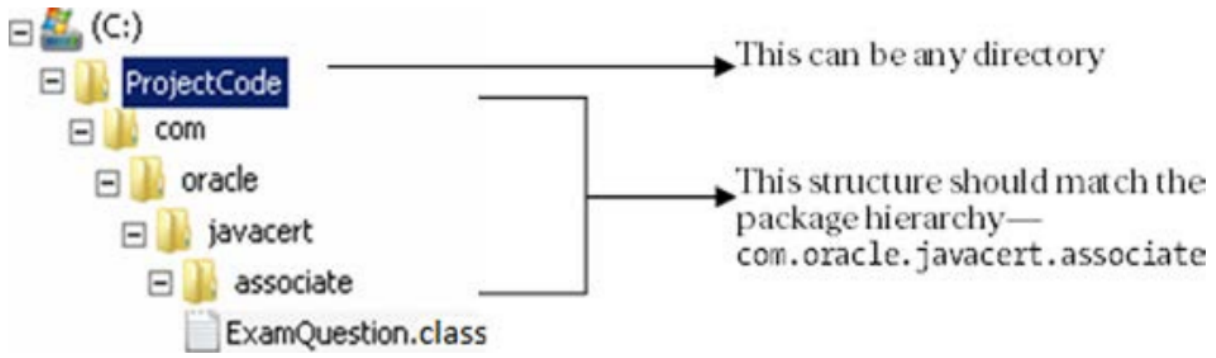
Package or subpackage name	Its meaning
`com`	Commercial. A couple of the commonly used three-letter package abbreviations are ▪ `gov`—for government bodies ▪ `edu`—for educational institutions
`oracle`	Name of the organization
`javacert`	Further categorization of the project at Oracle
`associate`	Further subcategorization of Java certification

- Su nomenclatura va de lo general a lo particular `com.oracle.javacert.associate`
- Siempre están escritos en minúsculas
- El paquete siempre se encuentra definido al inicio.
- Sólo puede existir 1 paquete por archivo
- El nombre cualificado de una clase o interface tiene como prefijo el nombre de su paquete y al final su nombre → `com.oracle.javacert.associate.ExamQuestion`

Estructura de directorio y jerarquia de paquetes

- La jerarquia definida en los paquetes debe coincidir con la jerarquia de directorios en los cuales se encuentran los archivos

Por ejemplo, para `com.oracle.javacert.associate.ExamQuestion` se tiene:



Utilizando nombres simples para importar

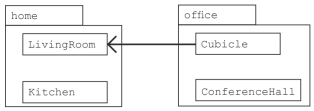
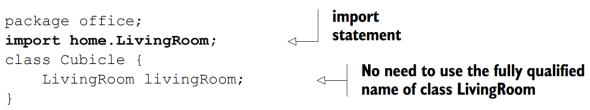


Figure 1.17 A UML representation of classes `LivingRoom` and `Cubicle`, defined in separate packages, with their associations

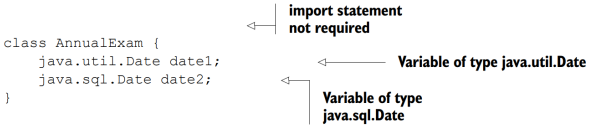
```
package office;
class Cubicle {
    home.LivingRoom livingRoom;
}
```

In the absence of an import statement, use the fully qualified name to access class LivingRoom.



Utilizando clases sin utilizar import

- Es posible utilizar clases empaquetadas sin `import` empleando su nombre completo cualificado.
- Para miembros del paquete `java.lang` las importaciones se realizan automáticamente.
- Es necesario emplear el nombre cualificado cuando se repite el nombre de las clases importadas




```
import java.util.Date;
import java.sql.Date;
class AnnualExam { }
```

Code to import classes with the same name from different packages won't compile.

- Utilizando el wildcard `*` puedes importar todas los miembros publicos, clases e interfaces del paquete

```
import certification.*;
class AnnualExam {
    ExamQuestion eq;
    MultipleChoice mc;
}
```

Imports all classes and interfaces from certification

Compiles OK

Also compiles OK

La sentencia import no importa todo el árbol de paquetes

- No se pueden importar clases de un subpaquete utilizando el wildcard `*`
- Para el ejemplo mostrado, la sentencia `import com.oracle.javacert.*;` únicamente importa la clase Schedule

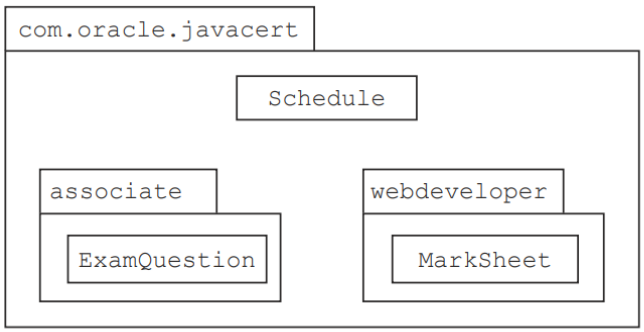


Figure 1.19 A UML representation of package `com.oracle.javacert` and its subpackages

Importando clases del paquete default

- Si no se incluye un paquete, las clases o interfaces pasan a formar parte del paquete por default sin nombre
- El paquete por deafult se agrega automáticamente
- Una clase en el paquete default no puede ser utilizada por una clase en un paquete nombrado
- Miembros de un paquete nombrado no pueden acceder a miembros en el paquete default

Importaciones static

Es posible importar un miembro estatico individual de una clase o todos sus miembros estaticos utilizando `import static`

- Para importar todos los miembros estaticos utiliza la marca `*`

Modificadores de acceso

Los modificadores de acceso controlan la accesibilidad de una clase o interfaz, incluyendo sus miembros. Usando el modificador apropiado se puede limitar el acceso.

- Los modificadores pueden ser aplicados a clases, interfaces y sus miembros.
- Variables locales y parámetros de métodos no pueden llevar modificador de acceso, de lo contrario no hay compilación
- Si no está definido el modificador, implícitamente se trata del `default` o también llamado *acceso de paquete*
- Los modificadores son `public` `protected` `default` `private`
- Las clases Top-level unicamene puede ser `public` y `default`

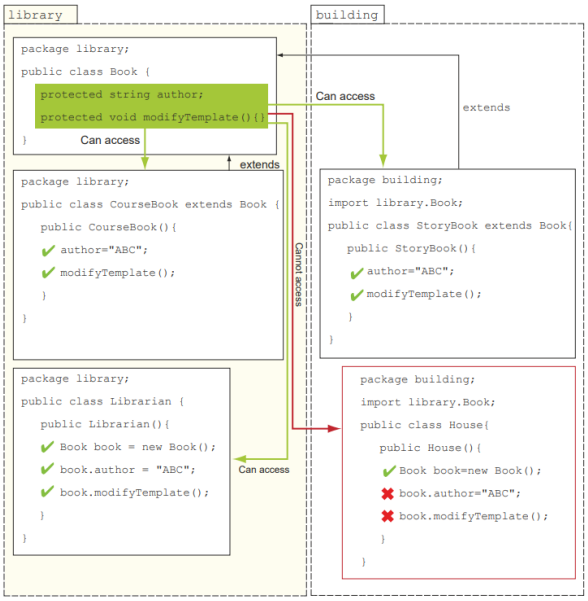
Modificador public

- Es el menos restrictivo, se da acceso a través de todos los paquetes

Modificador protected

Los miembros definidos como `protected` son accesibles por clases e interfaces definidas en el mismo paquete, todas las clases derivadas (incluso si se encuentran definidas en distintos paquetes)

- Puede verse como *package private +kids*



Modificador default

Los miembros defenidos con este modificador unicamente son accesibles desde el mismo paquete, por lo que se le puede referir como package-private, pues limita los miembros al paquete.

Modificador private

Es el modificador más restrictivo. Los miembros con este modificador unicamente son accesibles dentro del propio miembro, haciendo de lado incluso al propio paquete.

Modificadores Nonaccess

Abstract

Abstract class

Define un clase abstracta, incluso aunque ningún miembro sea abstracto.

- Una clase abstracta no puede ser instanciada
- Una clase concreta no puede definir elementos abstractos

Abstract interface

Una interfaz es una entidad abstracta por defecto, añadir la keyword es redundante.

Abstract method

Un método abstracto no tiene cuerpo, únicamente contiene la firma del método.

Abstract variables

Ninguna variable puede definirse como abstract, el código no compilará

Final

Final class

Una clase marcada como final no puede ser padre de otra clase

Final interface

Una interfaz no puede ser marcada como final, pues es abstracta y habría una contradicción

Final variable

Una variables marcada con final no puede ser reasignada en su referencia o valor, se convierte en una constante

- No tiene que estar inicializada obligatoriamente (permite una única inicialización)

Final method

Un método final es aquel que no puede ser sobreescrito por una clase derivada. Los hijos no puedes definir su propia versión

---

Static

Static variables

Las variables estáticas perteneces a la clase, no a la instancia.

- Una variables estática es compartida por todos los objetos de la clase
- La mejor forma de acceder a los miembros es a través de la clase y no del objeto

Static methods

Los métodos estáticos no están asociados a objetos, por lo que no pueden usar miembros de instancia, pero si los miembros de clase.

- Los miembros estáticos son heredados a clases derivadas
- Los miembros estáticos no se involucran en el polimorfismo de tiempo de ejecución
- No se puede sobreescribir miembros estáticos, sino redefinirlos
- Miembros de instancia pueden acceder a los miembros `static` , pues estos siempre existen independientemente de las instancias
- Los miembros static no pueden acceder a miembros de instancia
- Se pueden utilizar miembros estáticos a través de una variable de referencia con `null`

Características y comoponentes de Java

- Plataforma independiente → *"Write once, run anywhere"* | Java puede ser ejecutado en múltiples sistemas sin recompilación, el código de java se compila en un *bytecode* para posteriormente ser ejecutado en la JVM. Una JVM se encuentra instalada en el dispositivo e interpreta los bytecodes
- Orientado a objetos → Java emula la definición y comportamiento de objetos de la vida real.
- Abstracción → Mecanismo para modelar objetos de la vida real con sus atributos y comportamientos según sea necesario.
- Encapsulación → Permite controlar el nivel de acceso y modificación de los objetos o miembros
- Herencia → Mecanismo que permite una mayor abstracción y ahorro de código.
- Polimorfismo → Permite que las instancias exhiban múltiples comportamientos para el mismo método
- Tipado seguro → El tipo de dato de una variables no puede cambiar
- Manejo automático de memoria
- Multihilo y concurrencia
- Seguridad

- **Simuladores**
  - **Problema1**
  - **Problema2**