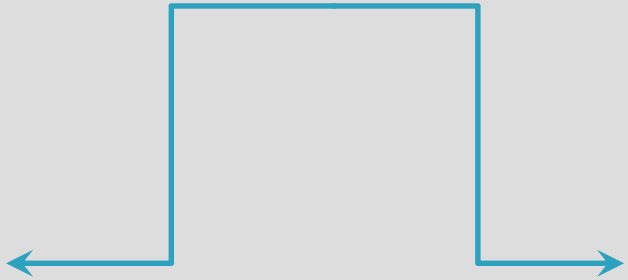
Programación Orientado a Objetos

Paradigmas de Lenguajes

TECNOLOGIA JAVA



Lenguaje de programación

de alto nivel

Plataforma

Ambiente de hardware y software donde corre un programa

Lenguaje de programación

- Simple
- Orientado a Objetos
- Distribuido
- Multihilo
- Dinámico
- Arquitectura Neutral
- Portable
- Alta performance
- Robusto
- Seguro

https://www.oracle.com/java/technologies/introduction-to-java.html#334

Objetivos de diseño

Simple: Los conceptos fundamentales de la tecnología Java se captan rápidamente. Se programa sin una extensa capacitación del programador. Los programadores pueden ser productivos desde el principio.

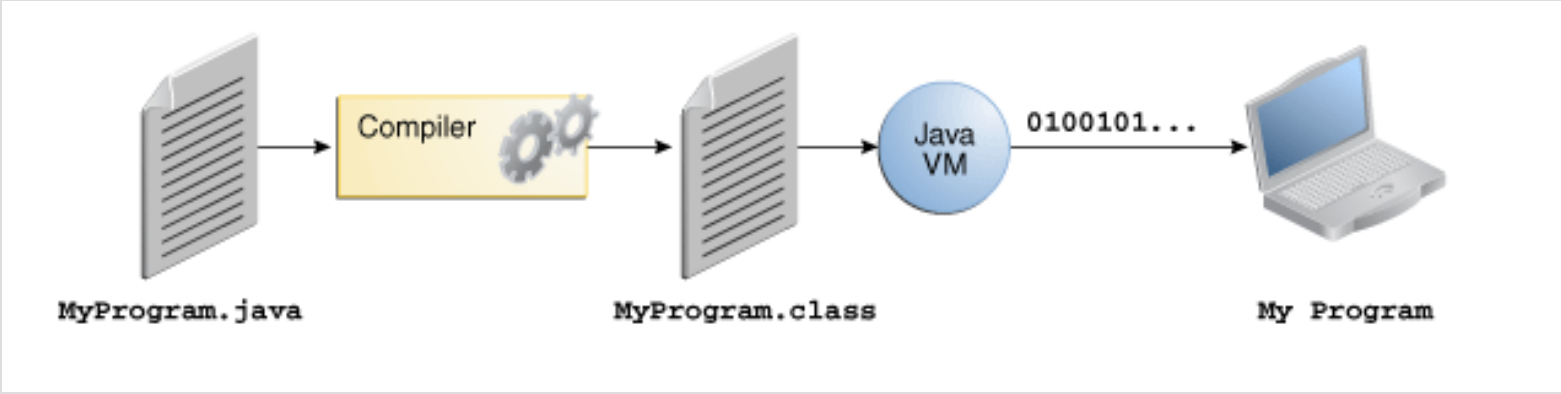
OO: Diseñado para orientarse a objetos desde cero. La tecnología Java proporciona una plataforma de desarrollo basada en objetos limpia y eficiente.

Distribuido: Las necesidades de los sistemas distribuidos basados en cliente-servidor coinciden con los paradigmas encapsulados de paso de mensajes del software basado en objetos. Para funcionar en entornos cada vez más complejos basados en red, los sistemas de programación deben adoptar conceptos orientados a objetos.

MultiHilo: Aplicaciones modernas basadas en redes, necesitan que el servidor haga varias cosas simultaneamente. La capacidad *multithreading* de la tecnologia java provee los medios para construir app con muchos hilos de ejecución concurrente.

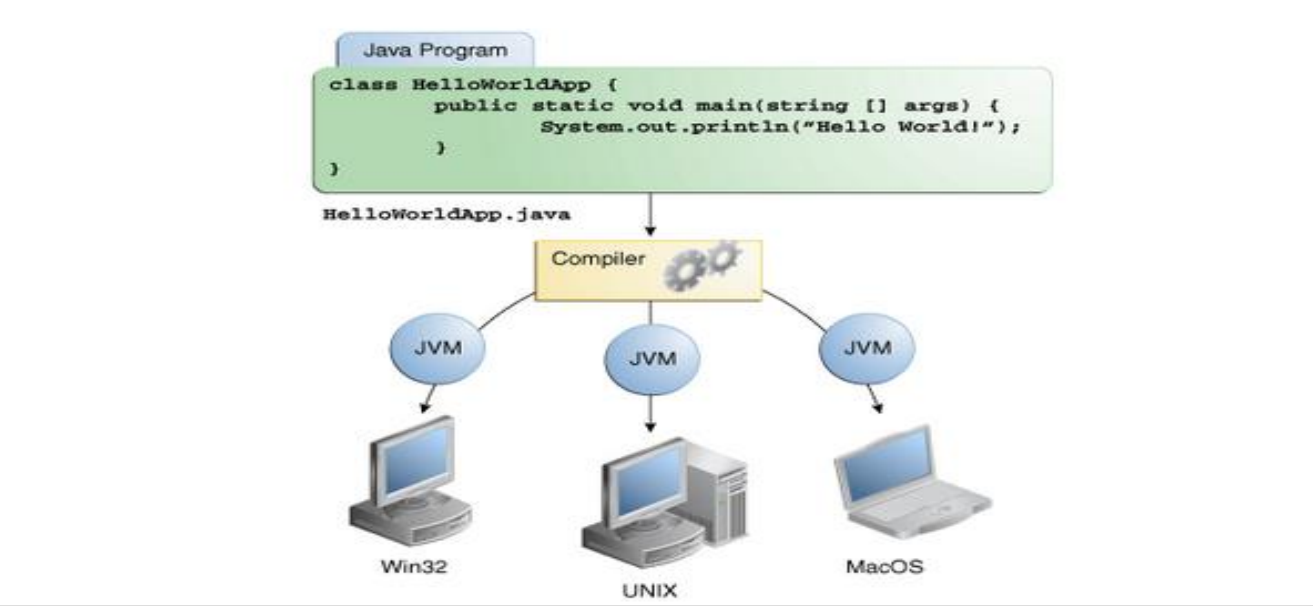
Dinámico: Si bien el compilador es estricto en su verificación estática en tiempo de ejecución, el lenguaje y el run-time son dinámicos en la etapa de vinculación. Las clases son vinculadas solo cuando se necesitan. Los nuevos módulos de código pueden vincularse a pedido desde una variedad de fuentes, incluso desde fuentes a través de una red.

Arquitectura Neutral: Las aplicaciones deben ser capaces de ejecutar sobre distintas arquitecturas, distintos sistemas operativos y deben interoperar con múltiples interfaces de lenguaje de programación. Para adaptarse, Java Compiler TM, genera bytecodes (el lenguaje de máquina de Java VM), un formato intermedio de arquitectura neutral diseñado para transportar código de manera eficiente a múltiples plataformas.



Portable: La máquina virtual Java se basa principalmente en la definición estándar de la especificación de la interfaz POSIX de una interfaz de sistema portátil. La implementación de la máquina virtual Java en nuevas arquitecturas es una tarea relativamente sencilla siempre que la plataforma objetivo cumpla con los requisitos básicos, como soporte para multihilo.

El mismo archivo .class puede correr en diferentes SO.



Alta performance: La plataforma Java logra un alto rendimiento. El recolector de basura automático se ejecuta como un subproceso de fondo con baja prioridad. Las aplicaciones con potencia de cálculo se pueden diseñar y reescribir en código de máquina nativo. En general, los usuarios perciben que las aplicaciones interactivas responden rápidamente aunque se interpreten.

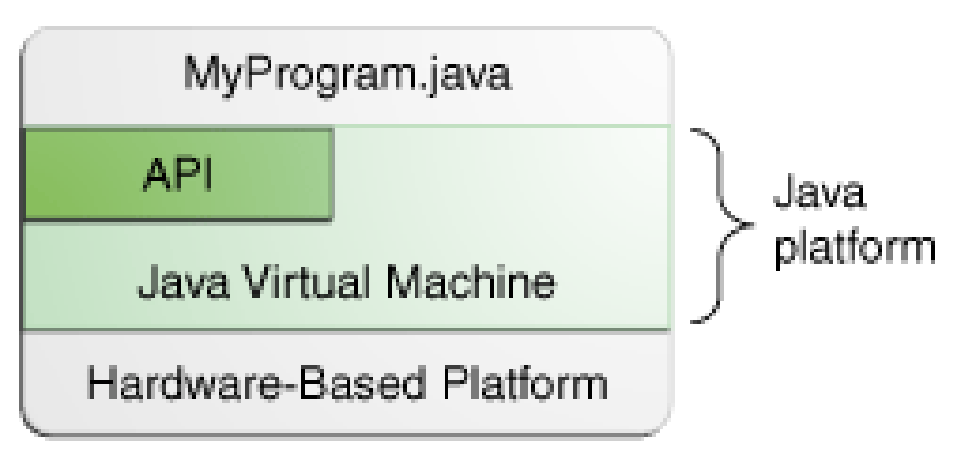
Robusto: Java está diseñado para crear software altamente confiable. Proporciona una extensa verificación en tiempo de compilación, seguida de una comprobación en tiempo de ejecución. El modelo de gestión de memoria es extremadamente simple: los objetos se crean con new. No hay tipos de datos de puntero definidos por el programador explícito, no hay aritmética de puntero y la recolección de basura es automática.

Seguro: Java está diseñada para operar en entornos distribuidos. Permite construir aplicaciones que no pueden ser invadidas desde afuera por instrucciones de código no autorizado que intenta ocultarse y crear virus o invadir sistemas de archivos.

PLATAFORMA JAVA

Java Virtual Machine

Java Application Programming Interface (API)



Java Platform Standard Edition | About the Java Technology (Th

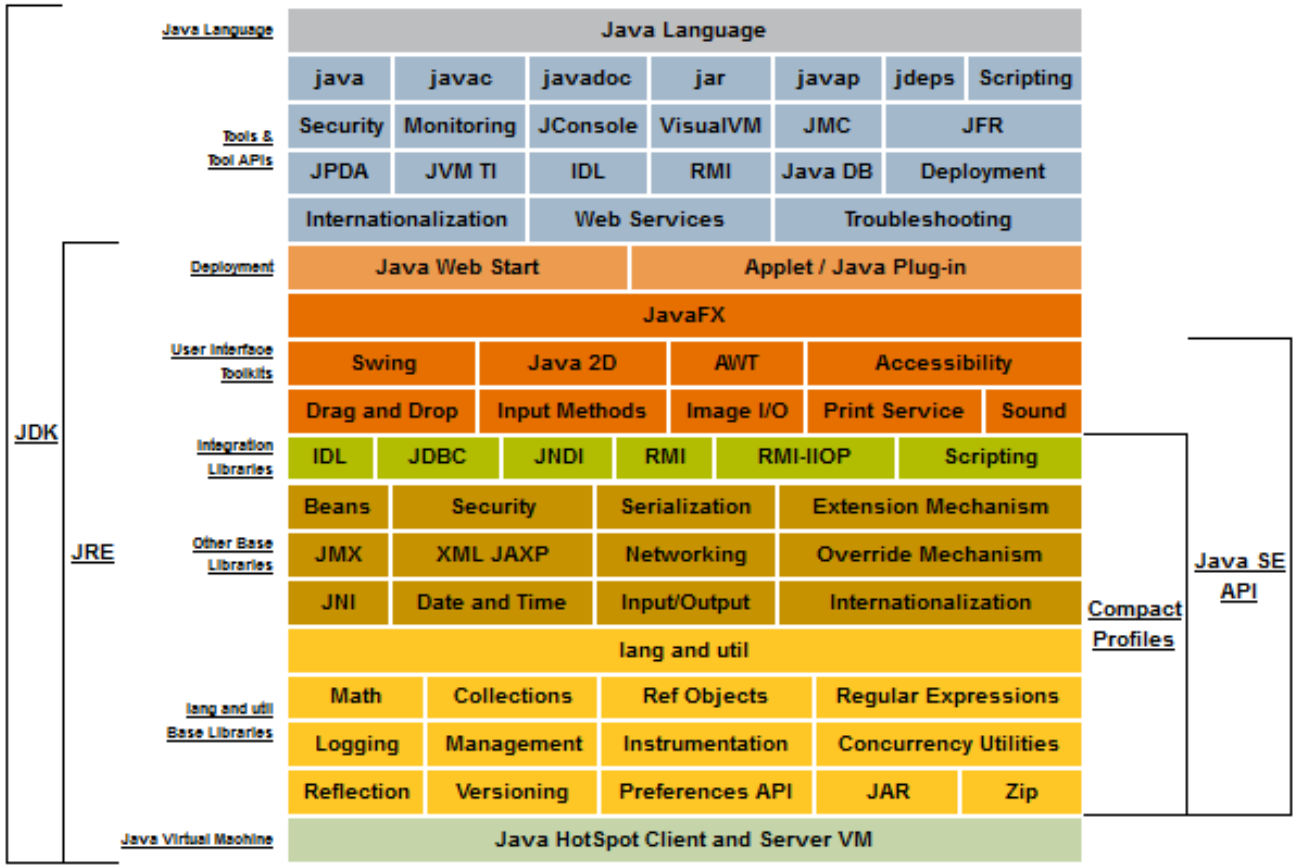
https://docs.oracle.com/javase/8/docs/index.html 80%

ORACLE Java Documentation

Oracle Technology Network Software Downloads Documentation

Java Platform Standard Edition 8 Documentation

Description of Java Conceptual Diagram



Revisión Modelo OO

Paradigmas de Lenguajes
3er Año LCC- Facultad CEFyN - UNSJ -

Pilares Programación OO

Abstracción proceso de exponer los detalles esenciales de una entidad, sin tener en cuenta los detalles irrelevantes, para reducir la complejidad.

Encapsulación proceso de agrupar datos y operaciones en los datos en una entidad.

Herencia se usa para derivar un nuevo tipo de un tipo existente, estableciendo así una relación padre-hijo.

Polimorfismo permite que una entidad adopte diferentes significados en diferentes contextos.

Java – ¿OO puro?

NO: Java conservado los tipos básicos de datos, byte, short, int, long, float, double, char y boolean que no son objetos.

SI: Toda definición de variables, constantes o funciones debe ser dentro de una clase.

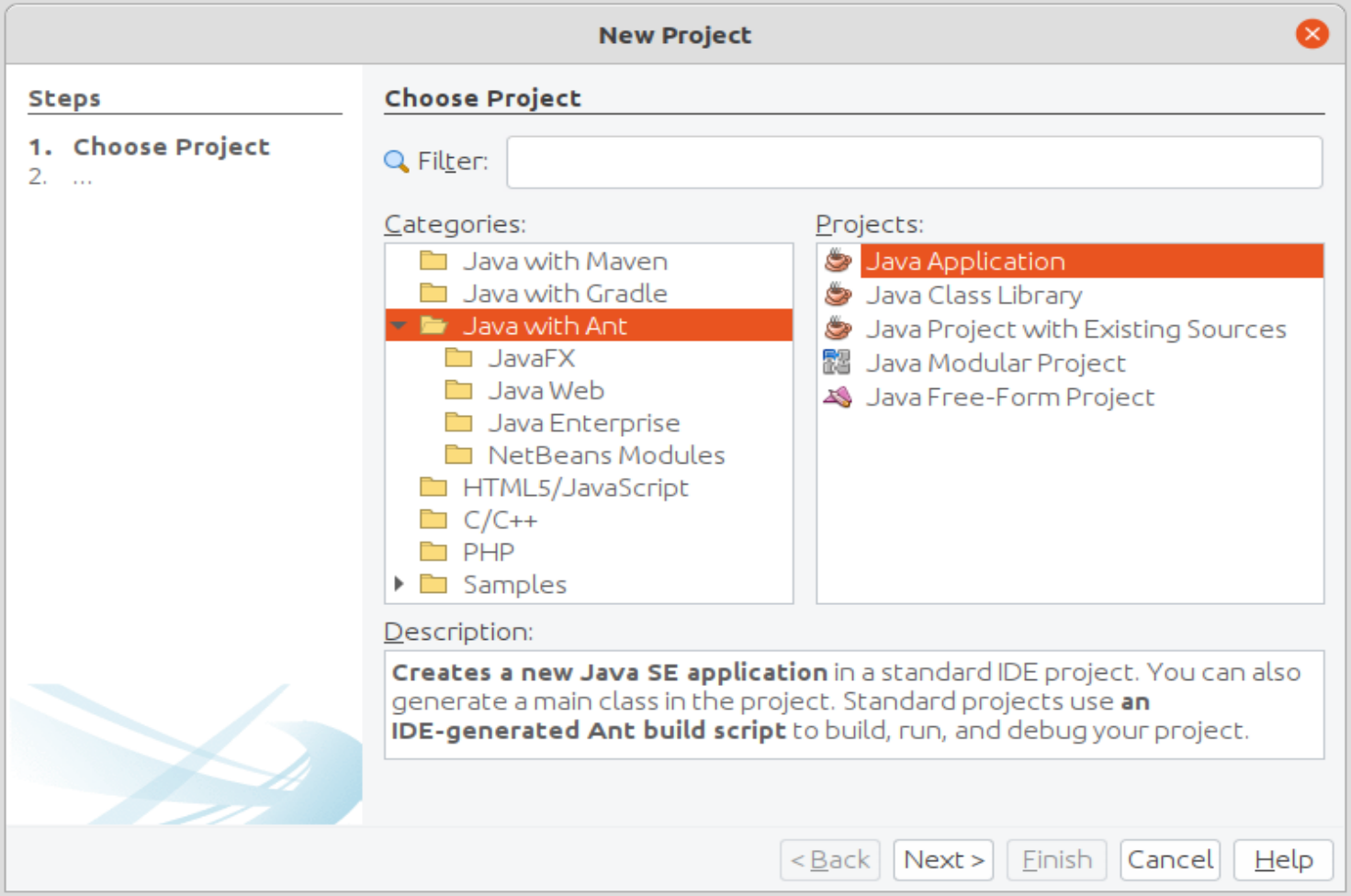
Conceptos básicos

Clase: es un blueprint que modela el estado y el comportamiento de un objeto del mundo real. Define: variables, constructor y métodos.

Objeto: es una instancia de una clase. Un objeto interactúa con otro por invocar métodos.

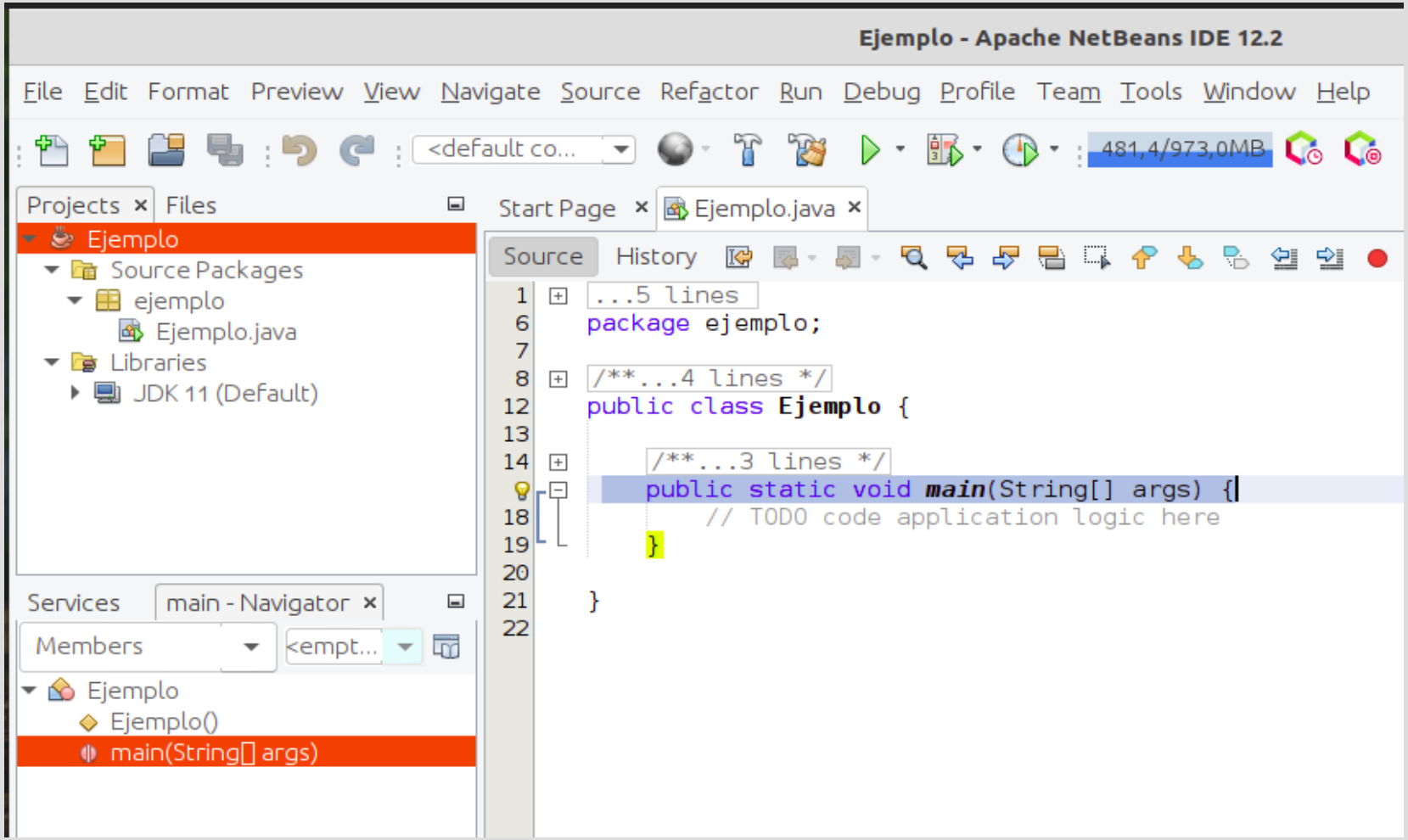
"A través de esta interacción, un programa puede llevar a cabo tareas, como implementar una GUI, ejecutar una animación o enviar y recibir información a través de una red"

Ejemplo:


New Project
Steps
1. Choose Project
2. ...
Choose Project
Filter:
Categories:
Java with Maven
Java with Gradle
Java with Ant
JavaFX
Java Web
Java Enterprise
NetBeans Modules
HTML5/JavaScript
C/C++
PHP
Samples
Projects:
Java Application
Java Class Library
Java Project with Existing Sources
Java Modular Project
Java Free-Form Project
Description:
Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use an IDE-generated Ant build script to build, run, and debug your project.
< Back
Next >
Finish
Cancel
Help

Entorno de desarrollo:

Netbeans https://netbeans.apache.org/



Java: Proyecto Ejemplo

1. **Paquete**
2. Librerías
3. Método main(String[] args)
4. Definir modelo – clases POJO
5. Definir manejador.

1. Paquetes

- Un paquete es un namespace que organiza un conjunto de clases e interfaces.
- Debido a que el software puede contener cientos o miles de clases individuales, tiene sentido mantener las cosas organizadas en paquetes.

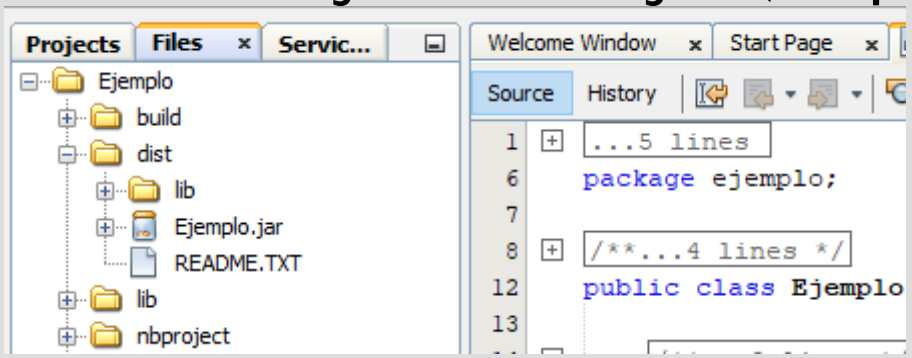
Documentación:

https://docs.oracle.com/javase/8/docs/api/index.html

https://docs.oracle.com/javase/9/docs/api/index.html?overview-summary.html

2. Librerías

- Contiene lo necesario para que funcione el proyecto.
- Las librerías se adjuntan al .jar (carpeta dist)



- Maven: (http://maven.apache.org/) es una herramienta que ahora se puede utilizar para crear y administrar cualquier proyecto basado en Java.

3.

```
public static void main(String[] args)
```

- **Punto de entrada principal para el launcher** (iniciador).
- Este método llama a **System.exit**.
- Parámetros: **args** – pueden ser cero o mas.
- Para correrlo desde la línea de comandos, agregar variable de sistema:

  JAVA_HOME = C:\Archivos de programa\Java**jdk1.8.0_65**

4. Ejemplo. Definir clases

Supongamos que se desea registrar las evaluaciones parciales de la materia.

```
package modelo;

public class Alumno {
    private String nombre;
    private Integer registro;
    private String carrera;
}
```

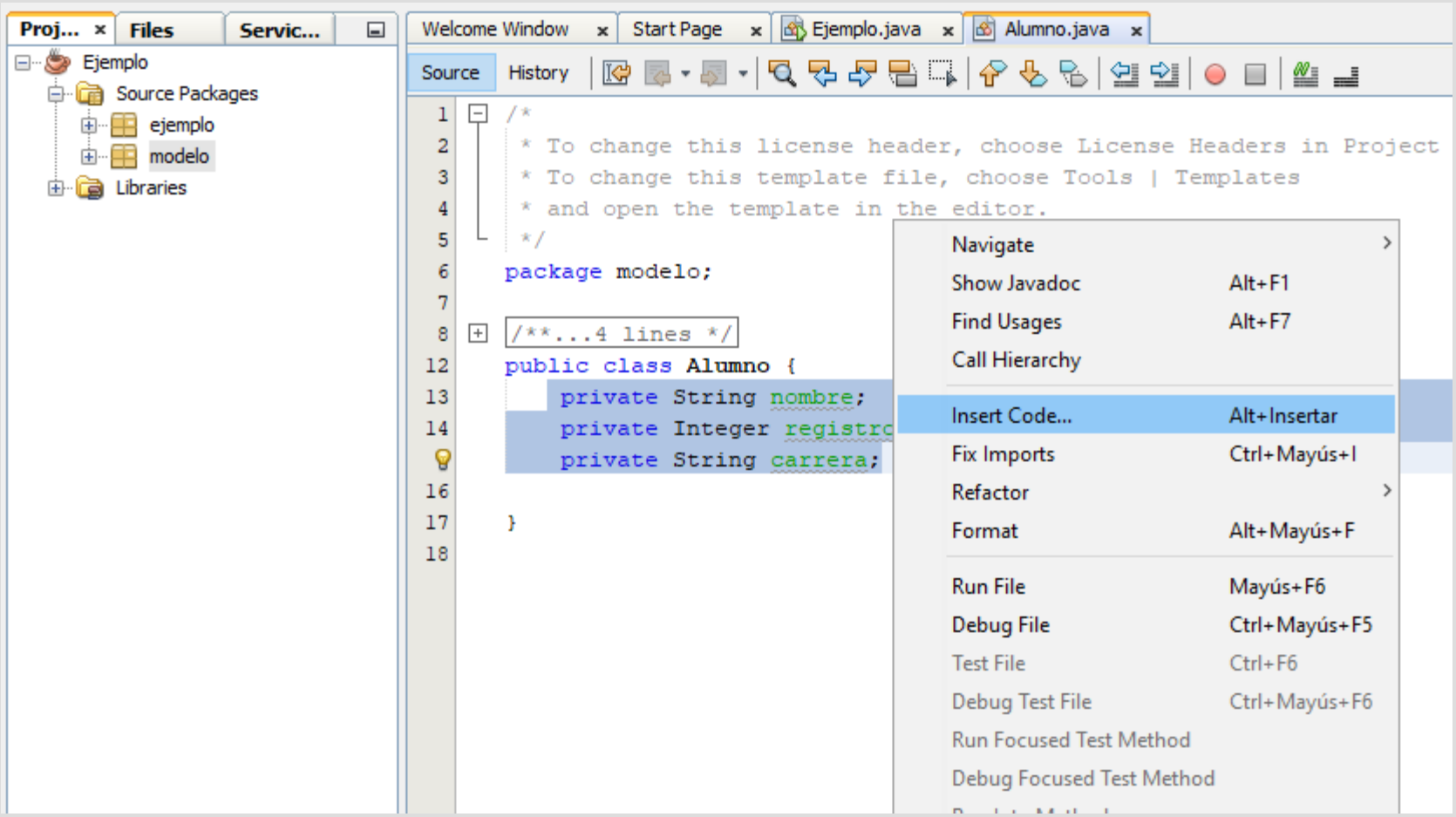
```
package modelo;

public class Parcial {
    private String nombre;
    private String nota;
}
```

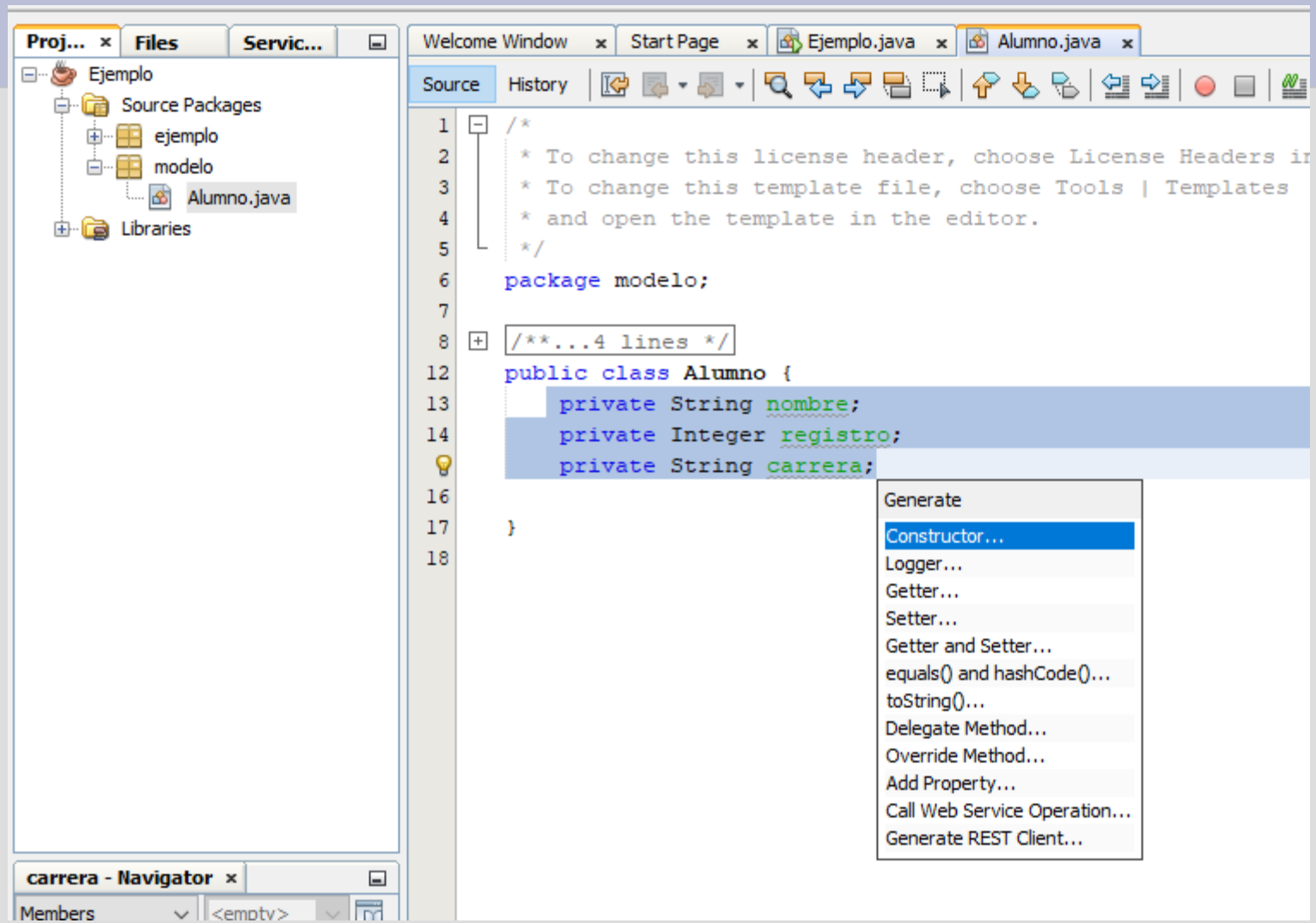
… clase “Object”

- La **Clase Object es la raíz en la jerarquía de herencias.**
- Toda clases en java tiene una superclase.
- Si no se especifica se asume que es la **clase Object** (declarada en el paquete java.lang).
- El paquete **java.lang** incluye las clases fundamentales de la biblioteca de Java. **Se importa por omisión.**

```
package modelo;
public class Alumno extends Object
{
    private String nombre;
    private Integer registro;
    private String carrera;
}
```

… Agregar código




Proj...
Files
Servic...
Ejemplo
Source Packages
ejemplo
modelo
Alumno.java
Libraries
carrera - Navigator
Members
<empty>
Welcome Window
Start Page
Ejemplo.java
Alumno.java
Source
History
1 /*
2 * To change this license header, choose License Headers in
3 * To change this template file, choose Tools | Templates
4 * and open the template in the editor.
5 */
6 package modelo;
7
8 /**...4 lines */
12 public class Alumno {
13 private String nombre;
14 private Integer registro;
private String carrera;
16
17 }
18
Generate
Constructor...
Logger...
Getter...
Setter...
Getter and Setter...
equals() and hashCode()...
toString()...
Delegate Method...
Override Method...
Add Property...
Call Web Service Operation...
Generate REST Client...

…constructor

"Método especial responsable de inicializar un nuevo objeto"

- Si el programador no lo define, se asigna **uno por defecto**.
- Tiene el mismo nombre de la clase, no se hereda, no retorna un valor, no puede ser declarado final, static ni abstract.
- Debe declararse como público para ser invocado desde cualquier lugar de la aplicación.
- La palabra clave **this()** se utiliza en un constructor para invocar al constructor de la misma clase cuya signatura coincide. Debe ser la primera línea de código en el constructor.
- La llamada **super()** en un constructor invoca al constructor del padre, y debe ser la primera sentencia del constructor o después de un this().

… constructor

```
package modelo;
public class Alumno {
    private String nombre;
    private Integer registro;
    private String carrera;

    public Alumno (String nombre, Integer registro, String carrera) {
            this.nombre = nombre;
            this.registro = registro;
            this.carrera = carrera;
    }
}
```

… (Python)

```
import modelo
class Alumno:
    def __init__ (self, nombre, registro, carrera=“”):
            self.__nombre = nombre
            self.__registro = registro
            self.__carrera = carrera
```

Características:

- No existe sobrecarga de métodos
- Los atributos inicializados en el constructor son de instancia:

  Alcance: __ Privado _ Protegido Publico
- Todas las clases heredan de **Object**
- Soporta herencia multiple
- **…**

```
package modelo;
public class Alumno {
    private String nombre;
    private Integer registro;
    private String carrera;
      public Alumno (String nombre, Integer registro, String carrera){
            this(nombre, registro);
            this.carrera = carrera;
        }
      public Alumno (String nombre, Integer registro) {
            this.nombre = nombre;
            this.registro = registro;
            this.carrera = “”;
      }
      }
```

```
package modelo;
import java.util.Scanner;  //  Link
public class Alumno {
    private String nombre;
    private Integer registro;
    private String carrera;
public Alumno (String nombre, Integer registro, String carrera){ … }
public Alumno (String nombre, Integer registro){ … }
public Alumno () {
        Scanner in = new Scanner(System.in);
        System.out.println("Cual es tu nombre? ");
        this.nombre = in.nextLine();   //String unaPalabra = in.next();
        System.out.println("Cual es tu registro ? ");
        if (in.hasNextInt()) {  //Chequea si hay un entero en la linea...
            this.registro = in.nextInt();  //float x = in.nextFloat();
        }
}}
```

… setter y getter

```
package modelo;
public class Alumno {
    private String nombre;
    private Integer registro;
    private String carrera;

    public Alumno (String nombre, Integer registro, String carrera) {..}

    public String getNombre() {
        return nombre;
     }
     public void setNombre(String nombre) {
        this.nombre = nombre;
     }
}
```

… toString()

```
package modelo;
public class Alumno {
    private String nombre;
    private Integer registro;
    private String carrera;
    public Alumno (String nombre, Integer registro, String carrera) {..}

   @Override
    public String toString() {
        return "Alumno{" + "nombre=" + nombre  + ", registro=" + registro  + '}';
    }
}
```

… equals() y hashCode()

```
package modelo;
public class Alumno {
    …
@Override
        public boolean equals(Object obj) {
         if (obj == null) { return false; }
         if (getClass() != obj.getClass()) { return false; }
         final Alumno other = (Alumno) obj;
         if (!Objects.equals(this.registro, other.registro)) { return false; }
                return true;
          }}
@Override
         public int hashCode() {
         int hash = 7;
         hash = 47 * hash + Objects.hashCode(this.registro);
         return hash;
    }
```

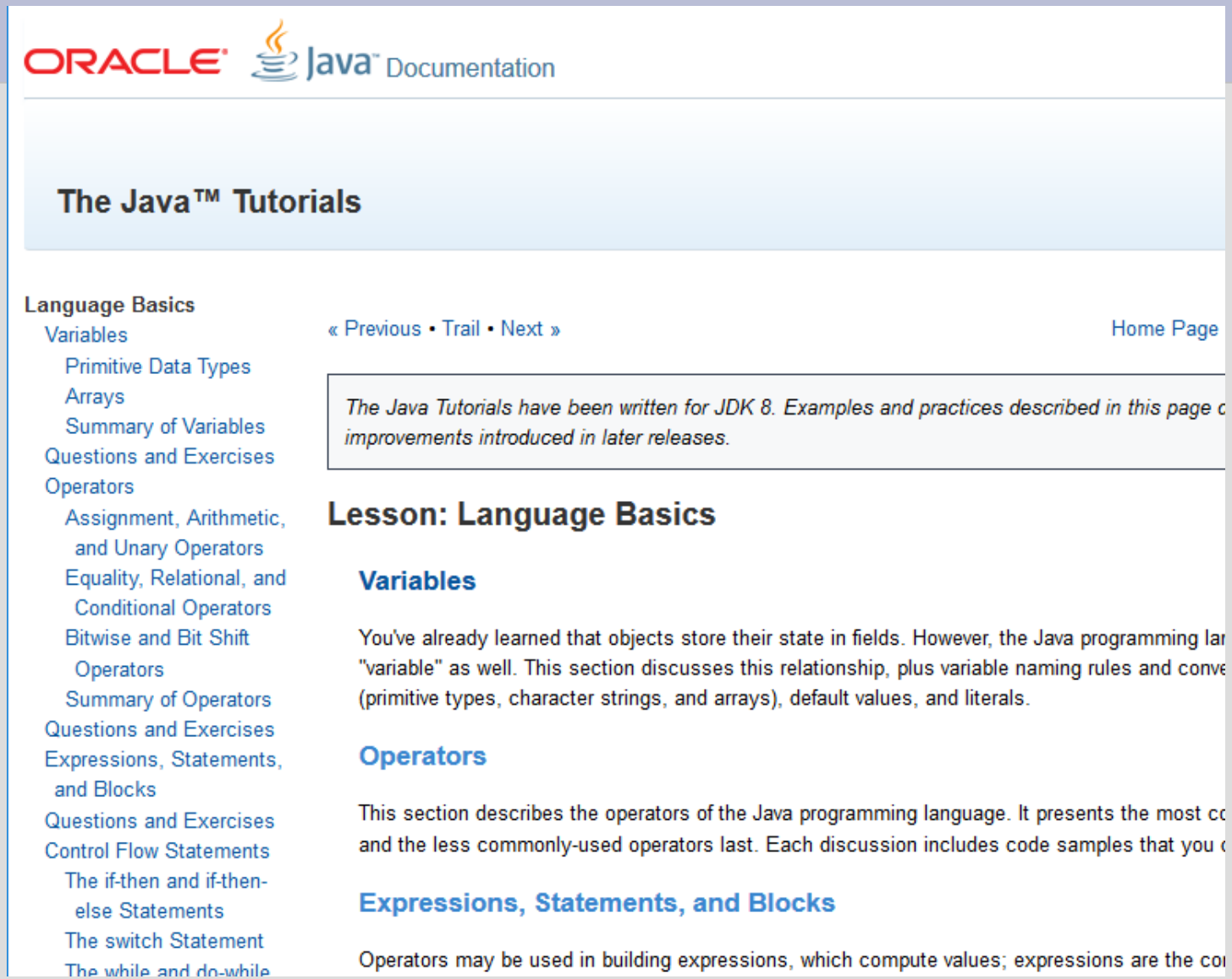
5. Definir Gestor/Manejador

```
package gestor;
import modelo.Alumno;
public class GestorAlumnos {
/*      private Alumno[] grAlumnos = new Alumno[40];   */
        private List<Alumno> grAlumnos ;
/*  Costructor   */
    public GestorAlumnos() {grAlumnos = new ArrayList<>();}
    public List<Alumno> getGrAlumnos() {…}
    public void setGrAlumnos(List<Alumno> grAlumnos) {.. }
    @Override
    public String toString() {..}
}
```

- Método vs mensaje
- Referencias de objetos
- Pasaje de parámetros
- Recolección automática de basura (*garbage collection),*
- Aspectos del lenguaje

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

Aspectos del Lenguaje



ORACLE Java Documentation

The Java™ Tutorials

Language Basics
- Variables
  - Primitive Data Types
  - Arrays
  - Summary of Variables
- Questions and Exercises
- Operators
  - Assignment, Arithmetic, and Unary Operators
  - Equality, Relational, and Conditional Operators
  - Bitwise and Bit Shift Operators
  - Summary of Operators
- Questions and Exercises
- Expressions, Statements, and Blocks
- Questions and Exercises
- Control Flow Statements
  - The if-then and if-then-else Statements
  - The switch Statement
  - The while and do-while

« Previous • Trail • Next » Home Page

The Java Tutorials have been written for JDK 8. Examples and practices described in this page c improvements introduced in later releases.

Lesson: Language Basics

Variables

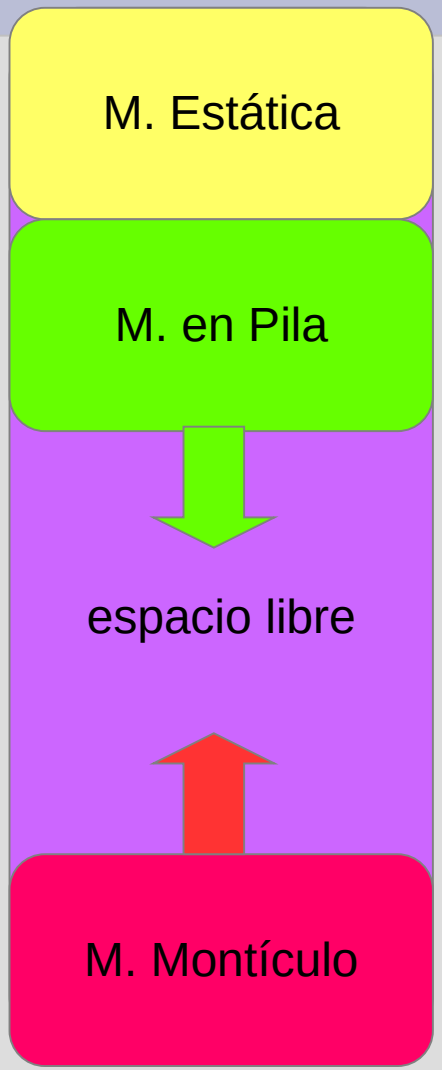
You've already learned that objects store their state in fields. However, the Java programming lar "variable" as well. This section discusses this relationship, plus variable naming rules and conve (primitive types, character strings, and arrays), default values, and literals.

Operators

This section describes the operators of the Java programming language. It presents the most co and the less commonly-used operators last. Each discussion includes code samples that you c

Expressions, Statements, and Blocks

Operators may be used in building expressions, which compute values; expressions are the co


M. Estática
M. en Pila
espacio libre
M. Montículo