# Proyecto Práctico TIC
# Tecnologías de la Información Empresarial

# Introducción a la Programación de Aplicaciones en Android


Juan Carlos Miranda

Universidad Paraguayo Alemana

San Lorenzo, Paraguay

E-mail: juancarlosmiranda81@gmail.com

## 1.1 Objetivos generales

- ✔ El módulo provee a los estudiantes los **conocimientos fundamentales** sobre los proyectos aplicados con énfasis en desarrollo web y mobile.

- ✔ El punto de vista de desarrollo **estará orientado a un proyecto práctico**, de manera que los estudiantes estén en condiciones de trabajar soluciones a problemas con estas tecnologías.

- ✔ El alumno tendrá la capacidad de **desarrollar aplicaciones para la plataforma Android** que incluyan los elementos más importantes de la Programación Orientada a Objetos.

## 1.1 Objetivos específicos

**Competencia académica y metodológica**

- ✔ Los estudiantes pueden pensar **en función de procesos y aplicar los conocimientos en ambientes prácticos** y mobile.
- ✔ Pueden **definir la arquitectura necesaria para un proyecto mobile**.
- ✔ Conocen los **métodos de modelación y desarrollo**, y pueden llevar a cabo proyectos de procesos de manera autónoma.
- ✔ Los estudiantes conocen **diferentes niveles de los procesos mobile** – desde el diseño hasta el nivel de funcionamiento.
- ✔ Conocen a profundidad la **metodología de desarrollo orientada a objetos**.

## 1.1 Objetivos específicos

**Competencia social y personal**

- ✔ Los estudiantes están **capacitados para manejar ejercicios** con procedimientos analíticos.
- ✔ Pueden **obtener de forma independiente informaciones** orientadas a planteamientos y aplicarlas para la resolución de problemas.

## 1.1 Evaluación y materiales

- ✔ Evaluación teórica **40%**.
- ✔ Presentación de un proyecto final **60%**.
- ✔ Se facilitan materiales con contenido teórico y referencias externas.
- ✔ Se acompaña con ejemplos de códigos fuentes para los casos tratados durante el curso.
- ✔ Los estudiantes desarrollarán o extenderán ejemplos.

## Instalación del entorno de trabajo

Esta unidad se centra en los **conceptos de Java**, no obstante, se da la libertad al estudiante de que elija el entorno de trabajo que más desee.

Existen varias alternativas **para compilar programas Java** entre las cuales se pueden mencionar:

- Apache Netbeans https://netbeans.apache.org/
- IntelliJ IDEA https://www.jetbrains.com/idea/
- Eclipse IDE https://www.eclipse.org/downloads/packages/installer

## Conceptos básicos de programación

- ✔ Java es un lenguaje **amplio en cuanto a sintaxis y funcionalidades**.
- ✔ La documentación oficial del lenguaje es **“ORACLE Java Documentation”** https://docs.oracle.com/javase/tutorial/java/
- ✔ Se hará un **repaso sobre los tópicos esenciales** para escribir programas.
- ✔ Esta unidad es un **paso previo a la programación de aplicaciones móviles**.

## Conceptos básicos de programación

A continuación se listan los tópicos que se abarcan:

| | |
|---|---|
| • Estructura de un programa Java.<br>• Tipos de datos en Java.<br>• Entrada y salida estándar.<br>• Manejo de salidas. | • Estructuras de control de flujo: condicionales y ciclos repetitivos.<br>• Manejo de arreglos y matrices.<br>• Colecciones (Collections).<br>• Manejo de archivos (opcional). |

## Estructura de un programa Java

```
package com.mycompany.intro_java;

public class MyFirstClassInJava {
    public static void main(String[] args) {
        System.out.println("My first program in Java!");
    }
}
```

Programa básico en Java **“MyFirstClassInJava.java”**.

Un programa Java comprende la declaración del **paquete, clase y método principal** que contiene las instrucciones del programa. A su vez se declaran los **argumentos externos**

https://introcs.cs.princeton.edu/java/11cheatsheet/

# Tipos de datos

| | |
|---|---|
| • int<br>• double<br>• boolean<br>• char<br>• String | • long<br>• short<br>• byte<br>• float |
| Tipos de datos incorporados en Java **“UserInputClass.java”**. | |

Los tipos de datos estándar del lenguaje Java, representan los tipos básicos **para crear elementos más complejos a partir de su combinación**

* Para mayor información ver **Tabla 1** del material de alumno.

# Tipos de datos

| A | | NOT A |
|---|---|---|
| False | False | False |
| False | True | False |

| A | B | A OR B |
|---|---|---|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

| A | B | A AND B |
|---|---|---|
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

| A | B | A XOR B |
|---|---|---|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | False |

# Entrada y salida estándar

```
Scanner sc01 = new Scanner(System.in);
System.out.print("Enter an integer number (eg. 1234):");
int intNumber;
intNumber = sc01.nextInt();

Scanner sc02 = new Scanner(System.in);
System.out.print("Enter a string data: ");
String stringData = “”;
stringData = sc02.nextLine();
```

Entrada de datos por teclado, salida por consola de texto. **“UserInputClass.java”**.

Entiéndase **entrada estándar** el uso del teclado para introducir datos y **salida estándar** el uso de una interfaz de texto en pantalla.

## Entrada y salida estándar

```
System.out.printf("Integer 10 places = | %14d |", intNumber);

System.out.printf("Double 10 places  = | %14.2f |", doubleNumber);

System.out.printf("String 10 places = | %.10s |", stringData);
```

Modificación de la salida estándar **“UserOutputClass.java”**.

La salida estándar, a su vez, **puede ser modificada para presentar distintos tipos de datos** según el formato que se requiera.

# Estructuras de control de flujo: condicionales

```
if (testValue < limit) {
    System.out.println("testValue < limit");
} else {
    System.out.println("Maybe ... testValue >= limit");
    if (testValue == limit) {
        System.out.println("testValue == limit");
    } else {
        System.out.println("testValue > limit");
    }
}
```

Condicional if-else. **“ControlFlowClass01.java”**.

Estructura **if – else** es utilizada para modificar el flujo del programa.

# Estructuras de control de flujo: condicionales

```
int testscore = 76;
char grade;
if (testscore >= 90) {
    grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) {
    grade = 'C';
} else if (testscore >= 60) {
    grade = 'D';
} else {
    grade = 'F';
}
System.out.println("Grade = " + grade);
```

Condicional if – else if. **“ControlFlowClass01.java”**.

Generalmente, la estructura **if – else if** suele ser utilizada para el manejo de opciones.

# Estructuras de control de flujo: condicionales

```
String message = null;
int optionSwitch = 5;
switch (optionSwitch){
    case 0: message = "Option 0";
        break;
    case 1: message = "Option 1";
        break;
    case 2: message = "Option 2";
        break;
    case 3: message = "Option 3";
        break;
    case 4: message = "Option 4";
        break;
    default: message = "Option DEFAULT";
         break;
}
```

Condicional switch. **“ControlFlowClass01.java”**.

La estructura **switch**, se aplica al control de opciones numéricas o de caracteres.

# Estructuras de control de flujo: ciclos repetitivos

```
int countWhile = 1;
int limitWhile = 10;

while (countWhile < limitWhile){
    System.out.println(countWhile);
    countWhile++;
}
```

Ciclo repetitivo while. **“ControlFlowClass02.java”**.

```
int countDoWhile = 1;
int limitDoWhile = 10;

do{
    System.out.println(countDoWhile);
    countDoWhile++;
} while(countDoWhile < limitDoWhile);
```

Ciclo repetitivo do … while. **“ControlFlowClass02.java”**.

La primera es el ciclo **while**, el cual ejecuta una condición de control como primera opción. La segunda, el el ciclo **do – while**, que realiza un control de flujo al final de las instrucciones.

# Estructuras de control de flujo: ciclos repetitivos

```
int countFor = 1;
int limitFor = 10;

for (countFor = 1; countFor < limitFor; countFor++){
    System.out.println(countFor);
}
```

Ciclo repetitivo for **“ControlFlowClass02.java”**.

Por último, la estructura **for**, con la cual se pueden procesar rangos de valores definidos.

Repositorio en Github con ejemplos:

https://github.com/juancarlosmiranda/java_recipes/

Un repaso hasta aquí.

Ejercicios propuestos en el material de alumno.

## Programación Orientada a Objetos (OOP)

- ✔ El paradigma de Programación Orientada a Objetos (OOP), **proporciona abstracción para diseñar software**.

- ✔ Un objeto es una construcción en la cual **se agrupan datos (propiedades)** y **procedimientos** para operar y acceder a esos datos (**métodos**).

- ✔ Este diseño favorece la **reutilización de componentes software** permitiendo escalar a diseños más complejos.

## Programación Orientada a Objetos (OOP)

Benficios de OOP:

- ✔ **Modularidad:** el código fuente de un objeto **se puede escribir y mantener independientemente** del código fuente de otros objetos.

- ✔ **Ocultación de información (encapsulamiento):** al **interactuar únicamente con los métodos de un objeto**, los detalles de su implementación interna permanecen ocultos al mundo exterior.

## Programación Orientada a Objetos (OOP)

Beneficios OOP:

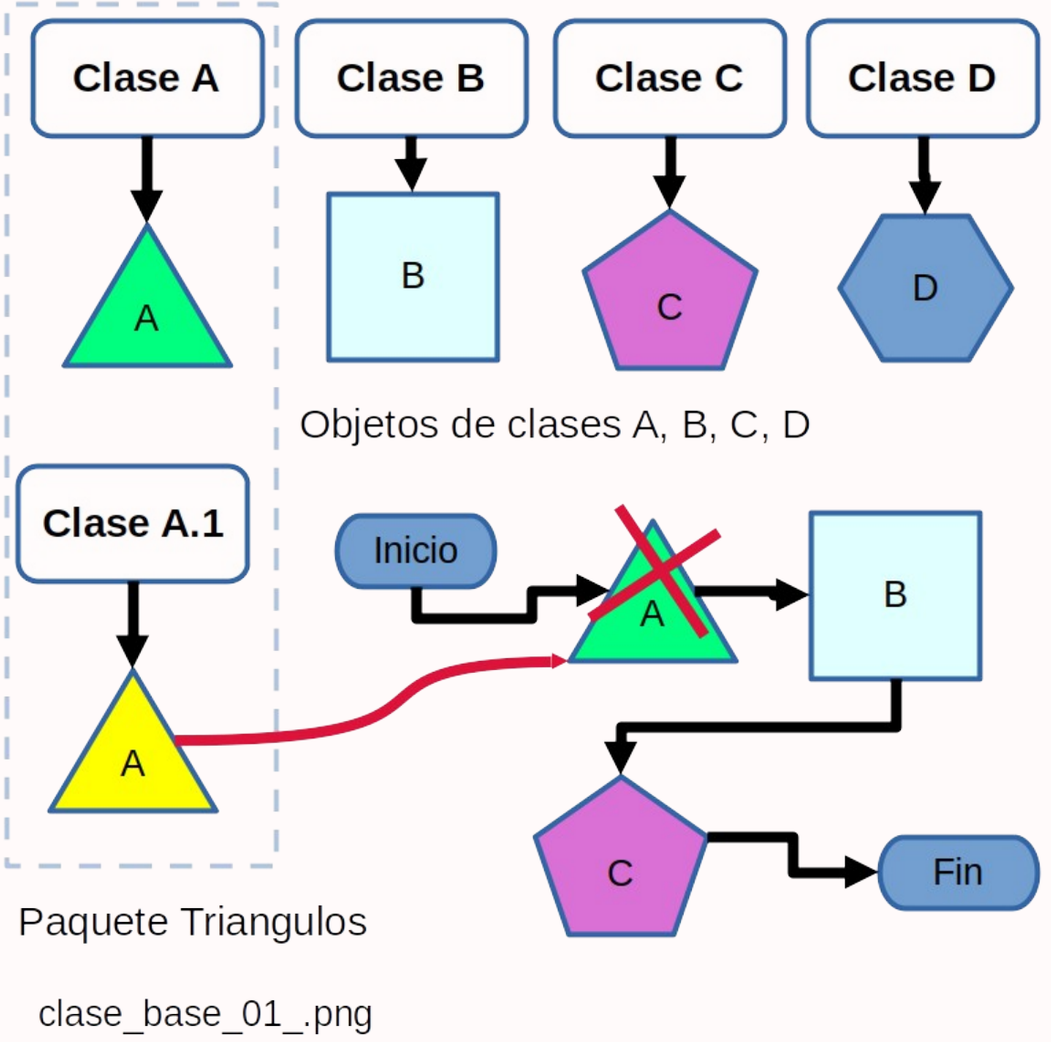
- ✔ **Reutilización de código:** si ya existe un objeto (tal vez escrito por otro desarrollador de software), se puede utilizar dicho objeto en un programa.

- ✔ **Facilidad de depuración:** si un objeto en particular resulta ser problemático, se puede eliminar de la aplicación y conectar un objeto diferente como reemplazo. Cada objeto puede ser depurado sin afectar a otros componentes.

## Programación Orientada a Objetos (OOP)

Cada objeto pertenece a un determinado tipo, **y este tipo se denomina clase**. Las clases, representan las **bases o plantillas** para crear objetos con comportamientos y propiedades especializadas.

En Java un paquete es un **espacio de nombres que permite organizar** clases e interfaces de manera conceptual

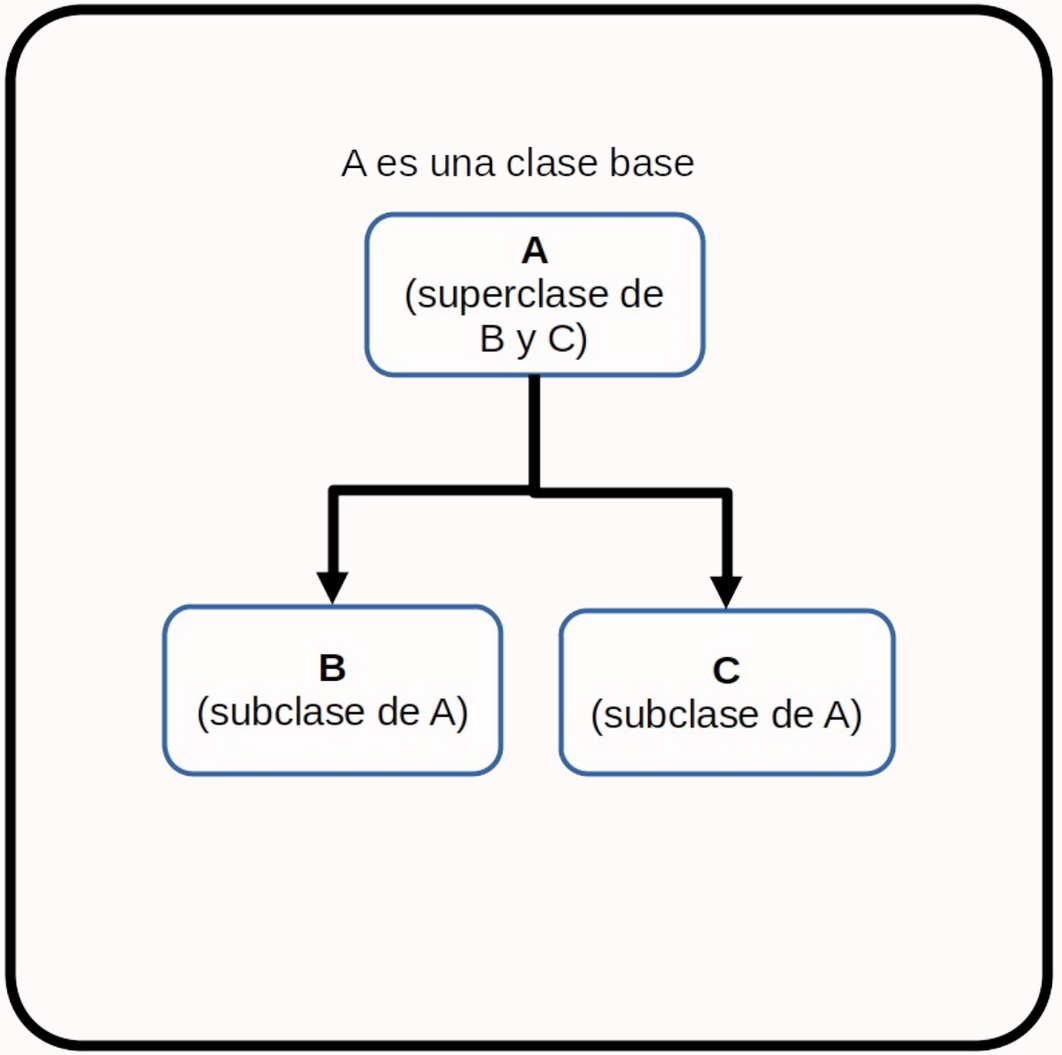



clase_base_01_.png

# Programación Orientada a Objetos (OOP)

En OOP, el **concepto de herencia** indica que las clases (tipos) pueden **heredar y transferir** propiedades y métodos.

- ✔ **Subclase (subtype / subclass)**, es la clase que hereda propiedades de otra clase.
- ✔ **Superclase (supertype / superclass)**, la clase que transfiere las propiedades a una subclase.
- ✔ **Clase base (base type)**, es una clase que posee varias subclases.



## Programación Orientada a Objetos (OOP)

Con las clases puedo hacer lo siguiente:

- ✔ Crear y destruir objetos.
- ✔ Comparar el tipo de clase.
- ✔ Crear clases heredando propiedades y métodos.
- ✔ Implementar nuevos métodos y propiedades dentro de las clases.
- ✔ Llamar a métodos de otras clases.
- ✔ Definir clases abstractas que servirán como base.
- ✔ Definir interfaces que servirán como base.

# Creación de Objetos

```
class Trivial {
    private long ctr;

    public void incr(){
        ctr++;
    }

    public long getIncr(){
        return ctr;
    }
}
```

Definición de clase Trivial y creación de un objeto. **“OopClassCreation01.java”.**

**“OopClassCreation01.java”**
**“OopClassCreation02.java”**

Un objeto es **una instancia de una clase**, en Java. Todas las variables internas a la clase son inicializadas por Java con 0 (cero) para tipos numéricos y null para String

```
public class OopClassCreation01 {
        public static void main(String[] args) {
            Trivial trivial = new Trivial();
            System.out.println(trivial.getIncr());
        }
}
```

Definición de clase Trivial y creación de un objeto. **“OopClassCreation01.java”.**

## Objetos, Herencia y Polimorfismo

- ✔ Se entiende por **“herencia”** cuando es posible extender clases (subclases) a partir de una clase existente (superclase).

- ✔ Se entiende por **“polimorfismo”**, cuando los subtipos (subclases) de una clase dada pueden ser asignados a una variable del tipo de clase base (superclase).

- ✔ Si existe una relación de herencia entre clases, un objeto puede cambiar de forma.

**“OopObjInheritance01.java”**
**“OopObjInheritance02.java”**
**“OopObjPolymorphism01.java”**

## Modificadores de acceso en Java

- ✔ Los modificadores **pueden cambiar el comportamiento de un objeto** definen el **alcance y la visibilidad** de las clases, métodos y propiedades (variables).

- ✔ Su funcionalidad cambia **dependiendo de la combinación** en las definiciones.

- ✔ Por defecto, en Java, si no se especifica un modificador de acceso, **las clases, métodos y propiedades son accesibles dentro del paquete que se han definido**.

**“OopObjInheritance01.java”**
**“OopObjInheritance02.java”**
**“OopObjPolymorphism01.java”**

## Modificadores de acceso en Java

- ✔ Los modificadores **pueden cambiar el comportamiento de un objeto** definen el **alcance y la visibilidad** de las clases, métodos y propiedades (variables).

- ✔ Su funcionalidad cambia **dependiendo de la combinación** en las definiciones.

- ✔ Por defecto, en Java, si no se especifica un modificador de acceso, **las clases, métodos y propiedades son accesibles dentro del paquete que se han definido**.

*** Ver Tabla 3 Modificadores de Acceso en Java.**

## Clases abstractas

- ✔ Las clases abstractas permiten **declarar conjuntos de métodos omitiendo la implementación** de código de los mismos.

- ✔ Las clases abstractas son **utilizadas como superclases**, **para luego implementar código específico en las subclases**.

- ✔ Son útiles cuando se desea implementar **un patrón de métodos a modo de plantilla**.

**“OopObjAbstract01.java”.**

# Clases abstractas

```
abstract class TemplateService {
   abstract void runService(); //abstract definition
   abstract void stopService();
   abstract void statusService();
   abstract void versionService();
}
class ServiceLinux extends TemplateService {
    void runService() {
       String s="s";
       System.out.println("SL - runService()");
    }
    void stopService() {
       System.out.println("SL - stopService()");
    }
}
```

Clase abstracta y la implementación “OopObjAbstract01.java”.

Representación de una clase abstracta (TemplateService) y una clase que implementa los métodos (ServiceLinux).

**“OopObjAbstract01.java”.**

# Interfaces

```
interface Animal {
  public void animalSound();
  public void sleep();
}

class Pig implements Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

Ejemplo de interface **“OopObjInterface01.java”**.

- ✔ Las interfaces permiten la declaración de tipos de datos **sin tener que definir la implementación del código**.
- ✔ Cada método definido en la interface **debe tener su implementación** correspondiente.

**“OopObjInterface01.java”.**

# Gestión de errores, manejo de excepciones

```
try{
    quotient01 = dividend01 / divisor01;
    reminder01 = dividend01 % divisor01;
} catch(Exception e){
System.out.println("I catched and error ->" + e.getMessage());
System.out.println("I catched and error ->" + e);

} finally {
        System.out.println("I always walk around here");
}
```

Ejemplo de uso try – catch - finally **“ErrorManagement01.java”**.

- ✔ Una excepción “Exception”, hace referencia **a una condición fuera de lo normal** que se produce en tiempo de ejecución de la aplicación.

**“ErrorManagement01.java”**
**“ErrorManagement02.java”**

*** Ver throw**

Repositorio en Github con ejemplos:

https://github.com/juancarlosmiranda/java_recipes/

Un repaso hasta aquí.

Ejercicios propuestos en el material de alumno.

Versión 1.0