



# Java

Introducción a Java

# Java



- No es OO puro
- No todo es un objeto
- Tipado estático
- Tipos “built-in” (int, long, etc.)
- Instrucciones de control
- Sintaxis similar a C

# Plataformas



- Java ME (*Micro Edition*)
- Java SE (*Standard Edition*)
- Java EE (*Enterprise Edition*)

## Aplicaciones a desarrollar

- Applets
- Aplicaciones desktop
  - interface gráfica
  - de consola
- Aplicaciones web

# Plataforma Java



- Las clases se escriben en un archivo .java con el mismo nombre de la clase
- El código se compila y se genera código intermedio: archivo binario .class
- El código intermedio es pasado a código de máquina en el equipo final por la JVM: Java Virtual Machine
- Para generar un programa ejecutable una clase tiene que implementar el método `main`.

# Hello, world!

Archivo de texto MyClass.java

Correspondencia entre archivo y clase

```
package ar.edu.itba.poo;
```

Toda clase pertenece a un "package"

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
    }  
}
```

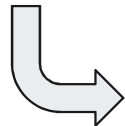
Argumentos por línea  
de comandos

Clase que no se puede instanciar con tres campos: in, err, out

# Classes abstractas

```
public abstract class Class1 {  
    public abstract Integer method1();  
}
```

```
public class Class2 extends Class1 {  
}
```



**Error:java: Class2 is not abstract and does not  
override abstract method method1() in Class1**

# Hello, world!

Versión gráfica con JavaFX

```
package ar.edu.itba.poo;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }
}
```

Para usar clases de otros packages se deben importar esas clases (o el package completo)

# Hello, world!

```
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Hello World!");
    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    btn.setOnAction(new MyAction());
    StackPane root = new StackPane();
    root.getChildren().add(btn);
    primaryStage.setScene(new Scene(root, 300, 250));
    primaryStage.show();
}

private class MyAction implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
}
```





# Hello, world!

## Clase anónima

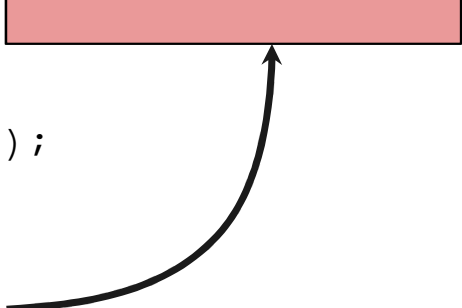


```
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Hello World!");
    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    btn.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Hello World!");
        }
    });
    StackPane root = new StackPane();
    root.getChildren().add(btn);
    primaryStage.setScene(new Scene(root, 300, 250));
    primaryStage.show();
}
```

# Hello, world!

lambda (funcional)

```
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Hello World!");
    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    btn.setOnAction(event -> System.out.println("Hello World!"));
    StackPane root = new StackPane();
    root.getChildren().add(btn);
    primaryStage.setScene(new Scene(root, 300, 250));
    primaryStage.show();
}
```



# Métodos y propiedades

```
public class Date {  
    private Integer year, month, day;           // inicializadas en null  
    private static String format;  
    private static final int MIN_YEAR = 1900;  
  
    public Date(int year, int month, int day) {  
        this.year = year;  
        this.month = month;  
        this.day = day;  
    }  
  
    public Date() {  
        year = 1970;  
        month = 1;  
        day = 1;  
    }  
}
```

# Polimorfismo



```
public static void setFormat(String format) {
    if ( isValid(format))
        Date.format = format;  // debo diferenciar ambos "format"
}

private static boolean isValid(String fmt) {
    return fmt != null && ...
}

private static boolean isValid(int year, int month, int day) {
    return ...
}

public Date clone() {
    return new Date(year, month, day);
}
```

# Validando en el constructor

```
public class Date {  
    int year=1970, month=1, day=1; // valores por defecto  
  
    public Date(int year, int month, int day) {  
        if ( !isValid(year, month, day)) {  
            throw new IllegalArgumentException("...");  
        }  
        this.year = year;  this.month = month;  this.day = day;  
    }  
  
    public Date() {  
    }  
    ...  
}
```

# Métodos: alcance

```
public class Foo {  
  
    Integer packageMethod() { ... }  
  
    public Integer publicMethod() { ... }  
  
    protected Integer protectedMethod() { ... }  
  
    private Integer privateMethod() { ... }  
  
}
```

# Ejemplo



```
public class Foo2 extends Foo {  
  
    @Override  
    public Integer publicMethod() { ... }    // OK  
  
    @Override  
    protected Integer protectedMethod() {  
        Integer n = privateMethod();        // ERROR  
    }  
}
```

# Métodos: modificadores

```
public abstract class Foo {  
    public static Integer staticMethod() { ... };  
  
    public final Integer finalMethod() { ... };  
  
    public abstract Integer abstractMethod();  
  
    public synchronized Integer syncMethod();  
}
```

```
public class Foo2 extends Foo {  
  
    public abstract void someMethod();  
}
```



# ==, equals

```
...
public int getYear() { return year; }

public int getMonth() { return month; }

public int getDay() { return day; }

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || ! (o instanceof Date) ) return false;

    Date date = (Date) o;

    // Si son int usar == si son Integer usar equals
    return year == date.year && month == date.month && day == date.day;
}
```

# ==, equals



```
Date d1 = new Date(2020, 11, 11);  
  
Date d2 = new Date(2020, 11, 11);  
  
System.out.println(d1==d2);           // false  
  
System.out.println(d1.equals(d2));    // true
```

# Herencia

```
public class DateTime extends Date {
    int hour, minute, second;

    public DateTime(int year, int month, int day, int hour, int min, int secs){
        super(year, month, day);
        ...
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!super.equals(o)) return false;
        ...
    }
}
```

# toString

```
public class Date {  
  
    @Override  
    public String toString() {  
        if (format == null)  
            return String.format("%02d/%02d/%d", day, month, year);  
        return String.format(format, day, month, year);  
    }  
  
}
```

```
public static void main(String args[]) {  
    Date d = new Date();  
    System.out.println("La fecha es " + d);  
}
```

La fecha es 01/01/1970

# Boxing, unboxing

Los tipos de datos primitivos no son objetos, no se deben instanciar para operar con ellos.  
Entre las clases wrapper no hay “casteos”

## Tipos de datos primitivos

byte  
short  
int  
long  
float  
double  
boolean  
char

## Clases wrapper

java.lang.Byte  
java.lang.Character  
java.lang.Short  
java.lang.Integer  
java.lang.Double  
...

**Boxing:** conversión de un tipo primitivo a una referencia

**Unboxing:** conversión de una referencia a su tipo primitivo

Automático a partir  
de JDK 1.5

# Boxing, unboxing: ejemplos

```
Integer n = 100;           // Equivalente a n = new Integer(100);
int n2 = n;                // Equivalente a n2 = n.intValue();

Integer n3 = new Integer(n);

Integer sum = n3 + n; // sum = new Integer(n3.intValue() + n.intValue());

n3 = null;                // OK
n2 = null;                // Error: Incompatible types

double x = n2;            // Error: Incompatible types
Double x2 = n;            // Error: Incompatible types
double real = 10;         // OK
```

# Boxing, unboxing: ejemplos



En el siguiente ejemplo, ¿qué método se invoca?

```
private static void method(double x) {...  
private static void method(Integer n) {...  
  
...  
  
method(10);
```

# Loops



```
while(boolean_expression)
    proposition
```

```
do
    proposition
while(boolean_expression);
```

```
for(initialization; boolean_expression; update)
    proposition
```



# Arrays



```
int v1[] = new int[10];
Integer []v2 = new Integer[10];
v1[2] = 10;
System.out.println(v1[3]);           // 0
System.out.println(v2[3]);           // null
System.out.println(v1[10]);           // Exception in thread "main"
                                     // java.lang.ArrayIndexOutOfBoundsException: 10
System.out.println(v2[1] + 5);        // Exception in thread "main"
                                     // java.lang.NullPointerException

System.out.println(v1.length);        // 10
System.out.println(v1.toString());    // [I@2626b418

Object[] v = {"Este array", 0, 1.5, "no tiene", 'x', "sentido" };
```

# Clase Arrays



Contiene métodos de clase para operar con vectores

```
static int binarySearch(Object[] a, Object key)

static int binarySearch(long[] a, long key)    // byte, char, int,
                                              // short, float, ...

static void sort(int[] a)

static boolean equals(long[] a, long[] a2)

static void fill(long[] a, int fromIndex, int toIndex, long val)

static void fill(float[] a, float val)
```

Ver <https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>

# Loops

```
Integer sum=0;  
for(int i=0; i < v1.length; i++) {  
    sum += v1[i];  
}
```

Ineficiente

```
int sum=0;  
for(int i=0; i < v1.length; i++) {  
    sum += v1[i];  
}
```

# Loops: for each

```
for(int value : v1) {  
    //  
}
```

```
Double avg = 0;  
for(int i : v1) {  
    avg += i;  
}  
avg /= v1.length;
```

Error:(line, col) java: incompatible  
types: int cannot be converted to  
java.lang.Double

```
double avg = 0;  
for(int i : v1) {  
    avg += i;  
}  
avg /= v1.length;
```

# Operadores



Categoría	Operador	Asociatividad
Postfix	() [] .	Izquierda a derecha
Unary	++ -- ! ~	Derecha a izquierda
Multiplicative	* /	Izquierda a derecha
Additive	+ -	Izquierda a derecha
Shift	>> <<	Izquierda a derecha
Relational	> >= < <=	Izquierda a derecha
Equality	== !=	Izquierda a derecha

# Operadores

Categoría	Operador	Asociatividad
Bitwise AND	&	Izquierda a derecha
Bitwise XOR	^	Izquierda a derecha
Bitwise OR		Izquierda a derecha
Logical AND	&&	Izquierda a derecha
Logical OR		Izquierda a derecha
Conditional	?:	Derecha a izquierda
Assignment	= += -= *= /= %= >>= <<= &= ^=  =	Derecha a izquierda

# Strings



Secuencia inmutable de caracteres

```
String s1 = "Hello";  
String s2 = "world";  
String s3 = s1.concat(", ").concat(s2);
```

```
String s1 = "Hello";  
String s2 = "world";  
String s3 = s1 + ", " + s2;
```

- Primero se crea un string con los caracteres de s1 seguido de “, “
- Luego se crea un nuevo string con el contenido del string creado seguido de los caracteres de s2
- s1 y s2 no cambian (no pueden cambiar)

# Strings

Leer una línea desde la entrada estándar y pasarla a mayúscula.

```
int ch;  
String s;  
  
while ((ch = System.in.read()) != -1 && ch != '\n')  
{  
    s += ch;  
}  
  
s.toUpperCase();
```

**Error:(122, 17) java: variable s might not have been initialized**

s no se modifica

Para una lista de los métodos de String ver  
[https://www.tutorialspoint.com/java/java\\_strings.htm](https://www.tutorialspoint.com/java/java_strings.htm)



# Strings



Leer una línea desde la entrada estándar y pasarla a mayúscula.

```
int ch;
String s= "";


while ((ch = System.in.read()) != -1 && ch != '\n')
{
    s += ch;
}

s = s.toUpperCase();
```

MUY ineficiente

Para el ejemplo anterior es más eficiente usar StringBuffer o StringBuilder

# Strings: constructores




```
String()  
String(byte[] bytes)  
String(byte[] bytes, Charset charset)  
String(byte[] bytes, int offset, int length)  
String(byte[] ascii, int hibyte, int offset, int count)  
String(char[] value)  
String(char[] value, int offset, int count)  
String(StringBuffer buffer)  
String(StringBuilder builder)
```

Para el detalle de constructores y métodos ver

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

# Strings: algunos métodos



```
char charAt(int index)
int compareTo(String anotherString)
int compareToIgnoreCase(String str)
boolean contains(CharSequence s)
boolean endsWith(String suffix)
boolean equalsIgnoreCase(String anotherString)
static String format(String format, Object... args)
byte[] getBytes()
int indexOf(int ch)
boolean isEmpty()
boolean matches(String regex)
String replace(char oldChar, char newChar)
String replaceAll(String regex, String replacement)
String[] split(String regex)
String trim()
String toUpperCase()
String toUpperCase(Locale locale)
static String valueOf(double d)
```

# Argumentos por línea de comandos

```
public static void main(String[] args) {  
    for(int i=0; i < args.length; i++) {  
        ...  
    }  
}
```

```
public static void main(String[] args) {  
    for(String s : args) {  
        ...  
    }  
}
```

# Excepciones



Todas las excepciones extienden directa o indirectamente de la clase Exception

- java.lang.**Object**
  - java.lang.**Throwable**
    - java.lang.**Error**
      - java.lang.**AssertionError**
      - ...
    - java.lang.**Exception**
      - java.lang.**CloneNotSupportedException**
      - java.lang.**InterruptedException**
      - java.lang.**ReflectiveOperationException**
        - java.lang.**ClassNotFoundException**
        - ...
      - java.lang.**RuntimeException**
        - java.lang.**ArithmeticException**
        - ...

# Excepciones

Si un método puede lanzar una excepción, debe indicarlo en su “*signature*”

```
public void method() throws DataAccessException {  
    if (...) {  
        throw new DataAccessException("...");  
    }  
}
```

El que invoque a *method()*  
debe manejar la excepción o  
estar dispuesto a propagarla

```
void method2() throws DataAccessException {  
  
    method();  
}
```

```
void method3() {  
    try {  
        method();  
    } catch (DataAccessException e)  
    {  
    }  
}
```

# Excepciones: capturar

```
try {  
    // Código que puede lanzar una excepcion  
} catch (Throwable1 e) {  
    ...  
} catch (Throwable2 e) { // Throwable1 no debe ser ancestro de Throwable2  
    ...  
}
```

```
try {  
    // Código que puede lanzar una excepcion  
  
} catch (Throwable1 | Throwable2 e) { // Throwable1 no debe ser  
    ancestro de Throwable2 y viceversa  
  
    ...  
}
```

# Excepciones: finally

En el bloque opcional “finally” se incluye código que se ejecuta tanto si se produce como si no se producen errores o excepciones

```
InputStream input = null;
try {
    input = FileUtils...
    ...
} catch (IOException ex) {
    // guardamos el error en un archivo de logs y lanzamos un
    RuntimeException
    throw new ...
} finally {
    if (input != null) {
        input.close();
    }
}
```

El bloque finally se ejecuta siempre,  
cuando finaliza el bloque try - catch

No se está considerando la posible excepción al cerrar el archivo



# Excepciones: finally

En el siguiente ejemplo nunca se ejecuta el bloque finally.

```
public class HelloGoodbye {  
    public static void main(String[] args) {  
        try {  
            System.out.println("Hello world");  
            System.exit(0);  
        } finally {  
            System.out.println("Goodbye world");  
        }  
    }  
}
```

System.exit() aborta  
la ejecución

Ejemplo extraído de Java Puzzlers

# Excepciones

Ejemplo: método para copiar dos archivos

```
static void copy(String src, String target) throws IOException {
    InputStream in = null;    OutputStream out = null;
    try {
        in = new FileInputStream(src);
        out = new FileOutputStream(target);
        int n;
        byte[] buffer = new byte[BUFF_SIZE];
        while ((n = in.read(buffer)) >= 0) out.write(buffer, 0, n);
    } finally {
        if (in != null) {
            try {        in.close();        } catch (IOException e) {        }
        }
        if (out != null) {
            try {        out.close();        } catch (IOException e) {        }
        }
    }
}
```

# Excepciones

Ejemplo: método para copiar dos archivos usando java.nio

```
static void copy(String src, String target) throws RuntimeException {  
  
    try {  
        Path in = Paths.get(src);  
        Path out = Paths.get(target);  
  
        Files.copy(in, out);  
    } catch (IOException e) {  
        throw new RuntimeException("Error al copiar");  
    } catch (InvalidPathException e) {  
        throw new RuntimeException("Alguno de los archivos no existe");  
    }  
}
```