Tipos de Datos

PRIMITIVOS

Tipos de Datos

```
byte b;
short s;
int i;
long l;
```

```
float f;
double d;
```

```
char c;
```

```
boolean bol;
```

Tipos de Datos

Tamaño en bits

```
byte b;  //8
short s; //16
int i;   //32
long l;  //64
```

```
float f;   //16
double d;  //32
```

```
char c;  //16
```

```
boolean bol;  //1
```

EL TAMAÑO SI IMPORTA

```
byte b;
int x=10;

b = x;
```

En un contenedor (byte) no se puede colocar algo más grande (int)

EL TAMAÑO SI IMPORTA

float f1 = 10; **10 por defecto es int**

float f2 = 10.0; **10.0 por defecto es double**

No se puede asignar un double en un float (float es más pequeño)

Tipos de Datos

Existen algunos sufijos para indicar explícitamente el tipo de dato

F : float
f : float

float f=10.0f;

d: double
D: double

L: long

CASTING

```
int x = 0;
short s;

s = x;

s=(short)x;

   float f=(float)10.0;
```

Si la información a guardar ocupa más bytes de los que tiene el nuevo contenedor, se truncan

VALORES POR DEFECTO

byte b;
short s;
int i;
long l;

0

float f;
double d;

0.0

char c;

'u0000' imprime un espacio en blanco

boolean bol;

false

Primitivos: Concatenación

Si sumo un *String con integer* el resultado será siempre un String

WRAPPERS

Tipos de Datos

Son CLASES que envuelven un valor primitivo.

WRAPPERS

Al ser clases, el valor por defecto de un atributo Wrapper es NULL

Byte b;
Short s;
Integer i;
Long l;

Float f;
Double d;

Boolean bol;


```
Integer i1=100;//5000
Integer i2=200;//5000

if(i1==i2){
   System.out.print("iguales");
}
```

Dos referencias son iguales con == si y solo si apuntan al mismo objeto. Por lo tanto si se quiere comparar los valores, NUNCA use == para comparar wrappers

```
if(i1.equals(i2)){
   System.out.print("iguales");
}
```

Se debe comparar con equals

Integer i1=100; //new Integer(100);

A los wrappers se les puede asignar el valor directamente, al ejecutar se crea el objeto

Wrappers: Unboxing

Los wrappers se pueden asignar a variables primitivas, el momento de ejecutar hacen **UNBOXING**, invocando al método **xxxValue();**

```
Integer i2 = 10;
int x= i2;
// i2.intValue();
```

```
Integer i3 = 10;
if(i3>10)          // unboxing

int z=5+i3;        // unboxing
```

Wrappers: Constructores

```
Integer i1=new Integer(10);
```

```
Integer i2=new Integer("20");
```

```
Float f2=new Float(20.0);
```

```
Float f3=new Float(20.0f);
```

```
Float f4=new Float("20.0");
```

CONSTRUCTORES WRAPPERS

Boolean b=false;

Boolean b1=new Boolean(true);

Boolean b2=new Boolean("Javita"); **false**

Boolean b3=new Boolean("True"); **true**

Boolean b4=new Boolean("true"); **true**

El constructor con parámetro String puede recibir CUALQUIER CADENA.
Resuelve true solo cuando la cadena es "true", sin importar mayúsculas o minúsculas.

MÉTODOS DE WRAPPERS

```
int x= Integer.parseInt(“10”);

double d=Double.parseDouble(“7.4”);

Integer a1=100;
int z = a1.intValue();

Float f1=100.0f;
int q = f1.floatValue();
```

Wrappers: Métodos Sobrecargados

```
public class DataType {
  public void m1(int x) { }
  public void m1(Integer x) { }
  public void m1(long x) { }
  public void m1(short x) { }
}
```

```
public static void main(String args[]){
  int x=100;
  DataType dt=new DataType();
  dt.m1(x);
}
```

Wrappers: Métodos Sobrecargados

1) Busca el tipo de **dato exacto**

2) Si no encuentra busca un **primitivo de mayor longitud**

3) Si no encuentra hace **BOXING** y busca su **wrapper correspondiente**

4) Si no encuentra el Wrapper específico busca una **clase padre del Wrapper**
   Padres: *Number, Object*

Wrappers: Métodos Sobrecargados

```
public class DataType {
        public void m1(int x) { }
        public void m1(Integer x) { }
        public void m1(long x) { }
        public void m1(short x) { }
}
```

```
public static void main(String args[]) {
        int x=100;
        DataType dt=new DataType();
        dt.m1(x);
}
```

Tipo de dato específico int

Wrappers: Métodos Sobrecargados

```
public class DataType {
        public void m1(int x) { }
        public void m1(Integer x) { }
        public void m1(long x) { }
        public void m1(short x) { }
}
```

```
public static void main(String args[]) {
        int x=100;
        DataType dt=new DataType();
        dt.m1(x);
}
```

Tipo de dato específico

Si no existe, busca un tipo de dato de mayor longitud long

Wrappers: Métodos Sobrecargados

```
public class DataType {
        public void m1(int x) { }
        public void m1(Integer x) { }
        public void m1(long x) { }
        public void m1(short x) { }
}
```

```
public static void main(String args[]) {
        int x=100;
        DataType dt=new DataType();
        dt.m1(x);
}
```

Tipo de dato específico

Si no existe, busca un tipo de dato de mayor longitud

Si no existe, se convierte en e

WRAPPER correspondiente

Integer

Wrappers: Métodos Sobrecargados

```
public class DataType {
    public void m1(int x) { }
    public void m1(Integer x) { }
    public void m1(long x) { }
    public void m1(short x) { }
}
```

1) Tipo de dato específico
2) Si no existe, busca un tipo de dato de mayor longitud
3) Si no existe, se convierte en el WRAPPER correspondiente
4) Si no existe busca la clase padre del Wrapper (Number-Object)

```
public static void main(String args[]) {
    int x=100;
    DataType dt=new DataType();
    dt.m1(x);
}
```