Conceptualización

Las conversiones de tipos consisten en transformar una variable de un tipo en otro. Las conversiones se pueden hacer hacia un tipo superior o hacia un tipo inferior. Si se utiliza una conversión hacia un tipo inferior, existe el riesgo de perder información. Por ejemplo, la conversión de un tipo **`double`** en un tipo **`long`** provocará la pérdida de la parte decimal del valor. Por ese motivo el compilador exige en estas situaciones que se indique de manera explícita que se desea realizar esta operación. Para ello, debe prefijar el elemento que desea convertir con el tipo que quiere obtener ubicándolo entre paréntesis.
Observe el siguiente ejemplo de código

```
float superficie;
superficie=2356.8f;
int aproximacion;
aproximacion=(int)superficie;
```

Se torna evidente que se pierde la parte decimal. La conversión explicita debe emplearse con cuidado, aunque muchas veces se recurre a ella a propósito como parte del objetivo que se persigue en el código desarrollado.
Las conversiones hacia un tipo superior no implican riesgo de perder información y por ese motivo el compilador de Java las realiza directamente mediante una simple asignación.

La tabla 1 resume las conversiones posibles y si deben ser explícitas (☹) o implícitas (☺).

	Tipo de datos a obtener						
Tipo de datos de origen	byte	short	int	long	float	double	char
byte		☺	☺	☺	☺	☺	☺
short	☹		☺	☺	☺	☺	☺
int	☹	☹		☺	☺	☺	☺
long	☹	☹	☹		☺	☺	☹
float	☹	☹	☹	☹		☺	☹
double	☹	☹	☹	☹	☹		☹
char	☹	☺	☺	☺	☺	☺	

Tabla 1. Conversión de tipo de datos posibles entre primitivos numéricos

Conversión hacia una cadena de caracteres

La conversión hacia el tipo cadena de caracteres son accesibles mediante la clase `String`. Esta clase posee el método de clase *valueOf*() para realizar la conversión de un valor de un tipo primitivo hacia una cadena de caracteres. Este método es del tipo sobrecargado, donde existe una versión del método para cada uno de los tipos primitivos. La ayuda del lenguaje para la clase `String` detalla en la Tabla 2.
Se puede observar que además de los primitivos se incluyen métodos sobrecargados para otros tipos de datos. Esto permite que mediante este método se pueda convertir cualquier tipo de datos (ya sea primitivo o una referencia) a una representación del tipo `String`.
Algunas veces no se necesitará recurrir a este método debido debido a otras características del propio tipo `String`. Por ejemplo, la concatenación de cadenas de caracteres mediante el operador **+** realiza la conversión de tipos a `String` antes de realizar la concatenación. Entonces estas dos porciones de código generan el mismo resultado:

```
double precioBruto;
precioBruto=152;
String recap;
recap="el importe del pedido es: " + precioBruto*1.16;
```

```
double precioBruto;
precioBruto=152;
String recap;
recap="el importe del pedido es: " +String.valueOf(precioBruto*1.16);
```

static String	valueOf(boolean b) Returns the string representation of the boolean argument.
static String	valueOf(char c) Returns the string representation of the char argument.
static String	valueOf(char[] data) Returns the string representation of the char array argument.
static String	valueOf(char[] data, int offset, int count) Returns the string representation of a specific subarray of the char array argument.
static String	valueOf(double d) Returns the string representation of the double argument.
static String	valueOf(float f) Returns the string representation of the float argument.
static String	valueOf(int i) Returns the string representation of the int argument.
static String	valueOf(long l) Returns the string representation of the long argument.
static String	valueOf(Object obj) Returns the string representation of the Object argument.

Tabla 2. El método *valueOf*() de la clase String

Conversión desde una cadena de caracteres hacia los primitivos

En muchas situaciones la información numérica está expresada en un formato `String`. Por este motivo para poder operar con estos valores previamente se deben convertir al tipo numérico adecuado.
Para lograr esto se pueden utilizar los wrpappers. Los wrappers son una representación equivalente de los primitivos en la forma de objetos. Cada tipo primitivo posee su propio wrapper

Tipo básico	Clase correspondiente
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Las clases Wrapper poseen un método *parse…*() mediante el cual convierte un String al tipo primitivo correspondiente. La siguiente tabla muestra la definición de este método para cada wrapper

Clase	Método
Byte	public static byte parseByte(String s)
Short	public static short parseShort(String s)
Integer	public static int parseInt(String s)
Long	public static long parseLong(String s)
Float	public static float parseFloat(String s)
Double	public static double parseDouble(String s)
Boolean	public static boolean parseBoolean(String s)

Por ejemplo:

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

    String valorEnteroString = "34";
    String valorFloatString = "1.56f";

    int valorEntero = Integer.parseInt(valorEnteroString);
    float valorFloat = Float.parseFloat(valorFloatString);

    System.out.println(valorEntero);
    System.out.println(valorFloat);
}
```

Para que esta conversión se pueda llevar a cabo la cadena de caracteres debe tener una forma adecuada para llevar a cabo la conversión, por ejemplo, este ejemplo generará una excepción debido a que ***valorEnteroString*** posee un valor que no podrá convertirse a un número:

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

    String valorEnteroString = "esto generará una execpción";
    String valorFloatString = "1.56";

    int valorEntero = Integer.parseInt(valorEnteroString);
    float valorFloat = Float.parseFloat(valorFloatString);

    System.out.println(valorEntero);
    System.out.println(valorFloat);
}
```

Por ese motivo, el compilador generará la siguiente excepción:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "as"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)
        at java.base/java.lang.Integer.parseInt(Integer.java:658)
        at java.base/java.lang.Integer.parseInt(Integer.java:776)
        at ar.edu.unju.fi.principal.Principal.main(Principal.java:15)
```

Conversión desde primitivos a wrappers y viceversa

Para asignar un primitivo a su correspondiente wrapper se recurre a su constructor o al mecanismo autoboxing. El autoboxing permite asignar un primitivo a una variable del tipo de wrapper correspondiente. Así las siguientes líneas son equivalentes:

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

El mecanismo inverso, denominado "unboxing", permite convertir automáticamente un tipo wrapper en un tipo básico. La variable entera del ejemplo anterior puede asignarse a una variable de tipo `int`.

```
int x;
x=entero;
```