6

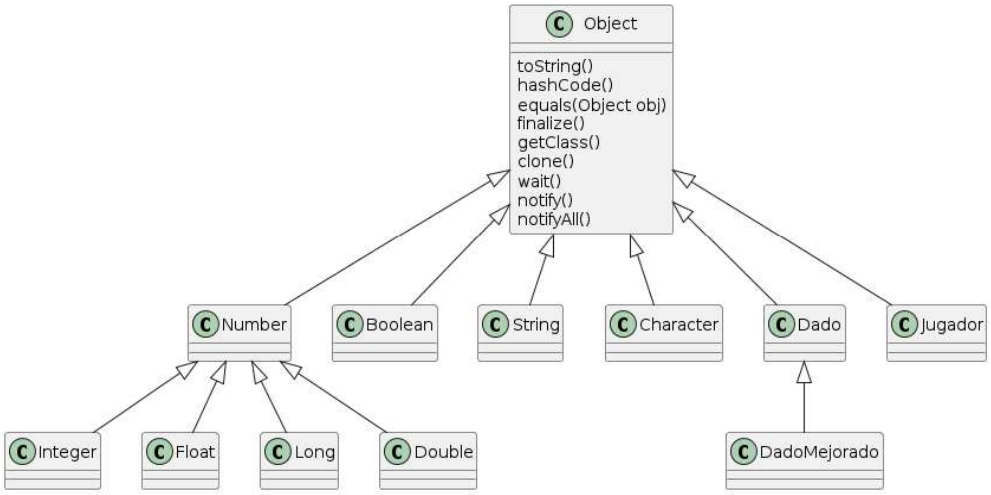
CAPITULO

Capítulo 6 - Programación Genérica

6. Programación Genérica

6.1. La Clase Object

La clase Object está presente en el paquete **java.lang**. Cada clase en Java se deriva directa o indirectamente de la clase Object. Si una clase no hereda de ninguna otra clase, entonces es una clase hija directa de Object y si hereda de otra clase, entonces hereda indirectamente de Object. Por lo tanto, los métodos de la clase Object están disponibles para todas las clases Java. Por tanto, la clase Object actúa como raíz de la jerarquía de herencia en cualquier programa Java.



La clase Object proporciona los siguientes métodos:

1. `toString()`: Este método devuelve una representación de cadena del objeto. Por defecto, devuelve una cadena que contiene el nombre de la clase del objeto y su dirección de memoria.

2. `equals(Object obj)`: Este método compara si el objeto actual es igual al objeto pasado como argumento. Por defecto, este método compara las direcciones de memoria de los objetos, pero es común sobrescribirlo en las clases derivadas para realizar una comparación significativa basada en el contenido de los objetos.

3. `hashCode()`: Devuelve un código hash único para el objeto. Este método es útil cuando se trabaja con colecciones basadas en hash, como HashMap, HashSet, etc.

4. `getClass()`: Devuelve el objeto Class que representa la clase del objeto. Esto puede ser útil para obtener información sobre la clase del objeto en tiempo de ejecución.

5. `notify()`, `notifyAll()`, `wait()`: Estos métodos son utilizados para la comunicación entre hilos. `notify()` despierta un hilo en espera que esté esperando en el objeto actual, `notifyAll()` despierta todos los hilos en espera y `wait()` hace que el hilo actual espere hasta que otro hilo lo notifique.

6. `finalize()`: Este método es llamado por el recolector de basura antes de liberar la memoria ocupada por el objeto. Es utilizado para realizar operaciones de limpieza o liberación de recursos antes de que el objeto sea destruido.

6.2. Genéricas

Como pudimos observar en la sección anterior, la clase **Object** es la superclase de todas las demás clases, entonces las referencias de tipo Object podrían usarse para referirse a cualquier objeto de cualquier tipo. Esta característica genera un problema de validación y control del tipo de dato respecto de los objetos. Las genéricas ayudan a gestionar este problema de seguridad sesgando los objetos hacia tipos concretos.

Podemos entender a las **genéricas** como una manera de tener tipos parametrizados en el programa, es decir, permitir que el tipo (entero, cadena, etc., y tipos definidos por el usuario) sean un parámetro para los métodos, clases e interfaces. Utilizando Genéricas, es posible crear clases que trabajen con diferentes tipos de datos. Una entidad como clase, interfaz o método que opera en un tipo parametrizado es una entidad genérica.

6.2.1. Tipos de genéricas en JAVA

- **Clases genéricas:** una clase genérica se implementa exactamente como una clase que no lo es, con la única diferencia de que la primera contiene una sección de parámetros de tipo. Puede haber más de unparámetro de tipo, separados por una coma. Las clases que aceptan uno o más parámetros se conocen como clases parametrizadas o tipos parametrizados.

- **Métodos genéricas:** Un método genérico tiene parámetros de Tipo que se citan por el tipo actual. Esto permite utilizar el método genérico de una manera más general. El compilador se ocupa del tipo de seguridad que permite a los programadores codificar fácilmente, ya que no tienen que realizar largas conversiones de tipos individuales.

6.2.2. Clases genéricas

Usamos <>para especificar tipos de parámetros en la creación de clases genéricas. Para crear objetos de una clase genérica, usamos la siguiente sintaxis.

```
class MiClase<T> {
        T atribA;
        MiClase(T atribA) { this.atribA = atribA; }
        public T getA() { return this.atribA; }
}

class Main {
  public static void main(String[] args)
  {
    MiClase<Integer> c1 = new MiClase<Integer>(15);
```

```
11      System.out.println(c1.getA());
12
13      MiClase<String> c2 = new MiClase<String>("Hola!");
14      System.out.println(c2.getObject());
15    }
16  }
```

6.2.3. Funciones genéricas

También podemos escribir funciones genéricas que se pueden llamar con diferentes tipos de argumentos según el tipo de argumentos pasados al método genérico. El compilador maneja cada método.

```
1  class Test {
2    static <T> void mostrarGenerico(T element)
3    {
4      System.out.println(element.getClass().getName()
5              + " = " + element);
6    }
7
8    public static void main(String[] args)
9    {
10     mostrarGenerico(11);
11
12     mostrarGenerico("Hola desde main!");
13
14     mostrarGenerico(1.0);
15   }
16 }
```

6.2.4. Ventajas de las genéricas

Los programas que usan Genéricas tienen muchos beneficios sobre el código no genérico.

1. **Reutilización de código:** podemos escribir un método/clase/interfaz una vez y usarlo para cualquier tipo que queramos.

2. **Seguridad de tipos:** las genéricas detectan errores que aparecen en tiempo de compilació. Es preferible identificar los problemas en código en tiempo de compilación en lugar de en tiempo de ejecución.

3. **Conversión de tipo:** La conversion un tipo de dato en cada operación de recuperación desde un conjunto de objetos o lista se puede tornar compleja. Si ya sabemos que nuestra lista solo contiene datos de tipo cadena, no necesitamos convertir cada vez.

4. **Implementación de algoritmos genéricos:** mediante el uso de genéricas, podemos implementar algoritmos que funcionan para diferentes tipos de objetos.