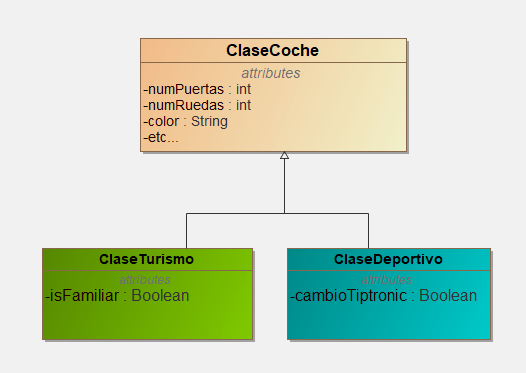
**La Herencia en Java**

Para entender que és la**herencia en java** la mejor forma es razonarlo desde el punto de vista de los objetos creados.

Cada objeto creado de una **Clase** determinada tiene el **estado** y **comportamiento** de esa Clase en particular. En ocasiones nos puede interesar crear distintos objetos con sus comportamientos específicos,  que tengan comportamientos  comunes a otros objetos. En este sentido entra en juego la **herencia entre Clases de Java**. Por ejemplo si nos fijamos en un vehículo a motor (un coche), distinguimos partes comunes a otros coches. Como pueden ser el «numero de ruedas», «el volante», «cambio de marchas», etc. Factores que son comunes en cada «objeto coche» que se fabrica.

Sin embargo especificando en cada objeto coche en particular, hay variaciones en sus características de fabricación. Estas pueden ser, si es  «Turismo o deportivo», «cilindrada», «la forma», que son **atributos** de cada tipo de **objeto coche**. Esto se puede ver mejor en la siguiente representación:



En este **diagrama de clases** de **[uml](https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado" \t "_blank)** vemos unarepresentación del concepto de**herencia en Java**. La Clase madre o Superclase ClaseCoche con sus atributos y dos Clases que heredan de ella. La Clase ClaseTurismo y la Clase ClaseDeportivo.

Ambas Clases tienen los atributos y métodos (si los hubiera) de la Clase ClaseCoche. Aparte cada una de estas clases hijas aporta los atributos y métodos que tengan. Si instanciamos un objeto de la clase «ClaseTurismo» , accederiamos a «isFamiliar» y al resto de miembros  de la SuperClase.

En  Java, a**cada clase se le permite tener una Superclase** , en cambio **una superclase puede tener infinitas subclases**.

## **¿Como hacer que una Clase herede de otra?**

Para crear una subclase hay que utilizar la [**palabra reservada extends**](https://codesitio.com/recursos-utiles-para-tu-web-o-blog/cursos/curso-de-java-palabras-reservadas-caracteres-especiales/)en la cabecera de la clase que hereda de la siguiente manera:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13 | class ClaseTurismo extends ClaseCoche {        // Todo el código de la clase aquí (atributos, constructores y otros métodos)    }      class ClaseDeportivo extends ClaseCoche {    // Todo el código de la clase aquí (atributos, constructores y otros métodos)    } |

En el **lenguaje Java** existe una jerarquia de Clases y encabezando está la **clase Object**,. A partir de ella se crea una estructura de niveles de herencias.

Una subclase hereda todos los *miembros* de la superclase a excepción de los **métodos constructores**, sin embargo puede accederse a ellos utilizando una invocación al mismo con la [palabra reservada super](https://codesitio.com/recursos-utiles-para-tu-web-o-blog/cursos/curso-de-java-palabras-reservadas-caracteres-especiales/), de la siguiente manera:

## **Ejemplo de uso de «super» en la herencia de Clases.**

Primeramente creamos una clase cualquiera, en este caso la llamaremos por ejemplo CCoche

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package cursojavaherencia;  public class CCoche {      //Campo o atributo de clase      private String color;        //Constructor de la Clase Madre      public CCoche(String color) {          this.color = color;      }        //Un método cualquiera      public void estaDisponible() {                   if(color=="azul" || color=="verde"){                      System.out.println("Su coche de color: " +color+ " esta disponible");                   }else{                      System.out.println("Su coche de color: " +color+ " no esta disponible");                 }         }  } |

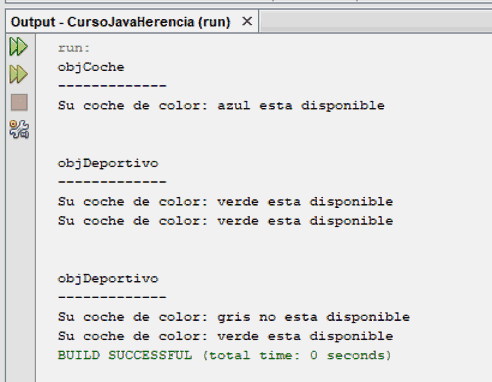
A continuación declaramos la clase CDeportivo que hereda de CCoche:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | package cursojavaherencia;  public class CDeportivo extends CCoche{      //Campo o atributo de clase        private String color;        // Constructor de la clase      public CDeportivo(String color) {          super(color); //Llamada al constructor de la superclase          this.color = color;      }        public void setColor(String color) {          this.color = color;      }         //Un método cualquiera      public void estaDisponible() {                   if(color=="azul" || color=="verde"){                      System.out.println("Su coche de color: " +color+ " esta disponible");                      super.estaDisponible();                   }else{                      System.out.println("Su coche de color: " +color+ " no esta disponible");                      super.estaDisponible();                 }         }  } |

Finalmente en la clase principal donde esta el método main creamos los objetos y lá logica del programa:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package cursojavaherencia;  public class CursoJavaHerencia {        public static void main(String[] args) {            // Creamos los objetos          CCoche objCoche = new CCoche("azul");          CDeportivo objDep = new CDeportivo("verde");             System.out.println("objCoche");           System.out.println("-------------");           objCoche.estaDisponible();  // Llamada a estaDisponible() de CCoche           System.out.println();           System.out.println();               System.out.println("objDeportivo");           System.out.println("-------------");           objDep.estaDisponible();   // Llamada a estaDisponible() de CDeportivo           System.out.println();           System.out.println();               System.out.println("objDeportivo");           System.out.println("-------------");           objDep.setColor("gris");   //Cambiamos valor del atributo "color" en CDeportivo           objDep.estaDisponible();   // Llamada a estaDisponible() de CDeportivo        }    } |

Si ejecutamos este programa obtendremos en consola el siguiente resultado:



## **Aclaraciones:**

* Primero: Creamos un objeto de cada clase «objCoche» y «objDeportivo» llamando al constructor  respectivo. Y pasamos como parámetro el valor de «color».
* Segundo: Desde «objCoche» invocamos el método «estaDisponible» de la clase «CCoche».
* Tercero: Desde «objDep» invocamos el método «estaDisponible» de la clase «CDeportivo». Este muestra primeramente un mensaje en consola y después llama al método «estaDisponible» de la «Superclase» CCoche. Pero si nos damos cuenta el atributo «color» ha variado de «azul»,  a «verde». Esto se debe a la llamada que se hace al constructor al crear objDep y pasar el parametro «color=verde».
* Cuarto: Se le asigna un nuevo valor al atributo «color» de la clase CDeportivo llamando a su método «setColor». Como vemos en consola solamente aplicariamos el cambio al atributo miembro de su misma clase y no a la Superclase.

## **Conclusiones sobre la herencia en Java.**

* Una**subclase hereda  los miembros *públicos* y *protegidos*** de la «Superclase» si  está en el mismo paquete.
* Los **miembros *privados* solamente se heredan si están en mismo paquete.**
* Pueden utilizarse **mismo nombre** para designar miembros tanto en la «superclase» como la subclase, esto responde al concepto de **ocultación de miembros de la superclase**.
* **Se pueden crear tantos atributos o métodos en la subclase como consideremos** oportuno y a su vez poder utilizar todos los miembros de la «Superclase».
* Tal y como hemos visto en el programa de ejemplo anterior, podemos invocar al constructor o a cualquier método de la «Superclase» utilizando la [palabra reservada «super»](https://codesitio.com/recursos-utiles-para-tu-web-o-blog/cursos/curso-de-java-palabras-reservadas-caracteres-especiales/), con la salvedad de que si invocamos al**constructor de la Superclase** desde el **constructor de la subclase** la llamada con «super» necesariamente debe estar situada en la primera linea del método constructor de la subclase.

# Polimorfismo

Un buen ejemplo de **Polimorfismo** lo econtramos en objetos contenedores de cualquier liquido, por ejemplo una botella, un vaso, etc. Según el objeto donde esté contenido, la forma del agua cambiará o se adaptará al recipiente contenedor.  En Java pasa algo parecido, **el valor retornado de un método varia según el objeto que le invoque**. Esto se llama **selección en tiempo de ejecución de métodos**y es una **manifestación de Polimorfismo en Java.**

La mejor forma de entender el **concepto de polimorfismo en Java** es con ejemplos prácticos. Por ejemplo si creamos una clase abstracta con un **atributo**, un **método normal** y otro **abstracto**:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package polimorfismo;  abstract class Clase\_Abstrac {      int num = 10;        //Método normal      int calcula(){return num\*num;}        //Método Abstracto      abstract int calcula2();  } |

Y dos clases o subclases que hereden de esta clase abstracta. Esta es la primera:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package polimorfismo;  public class Clase\_A extends Clase\_Abstrac{  int numA;    //Implementación del método abstracto  @Override      int calcula2() {        numA = calcula();        return numA + 10;      }  } |

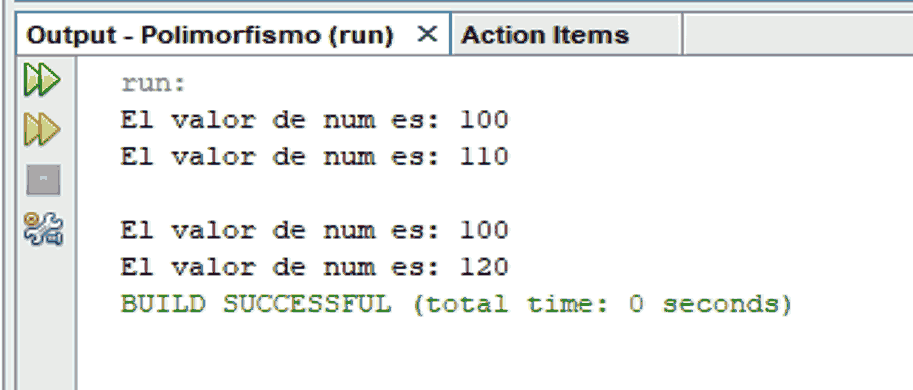
Y esta la segunda:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package polimorfismo;  public class Clase\_B extends Clase\_Abstrac{  int numB;        //Implementación del método abstracto      @Override      int calcula2() {        numB = calcula();        return numB + 20;      }  } |

Si creamos objetos de ambas clases y hacemos la llamada a distintos métodos desde la clase principal que contiene el método main:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package polimorfismo;  public class Polimorfismo {      public static void main(String[] args) {          Clase\_A objA = new Clase\_A();          Clase\_B objB = new Clase\_B();            System.out.println("El valor de num es: "+objA.calcula());          System.out.println("El valor de num es: "+objA.calcula2());          System.out.println();          System.out.println("El valor de num es: "+objB.calcula());          System.out.println("El valor de num es: "+objB.calcula2());      }  } |

Nos da el siguiente resultado en consola:



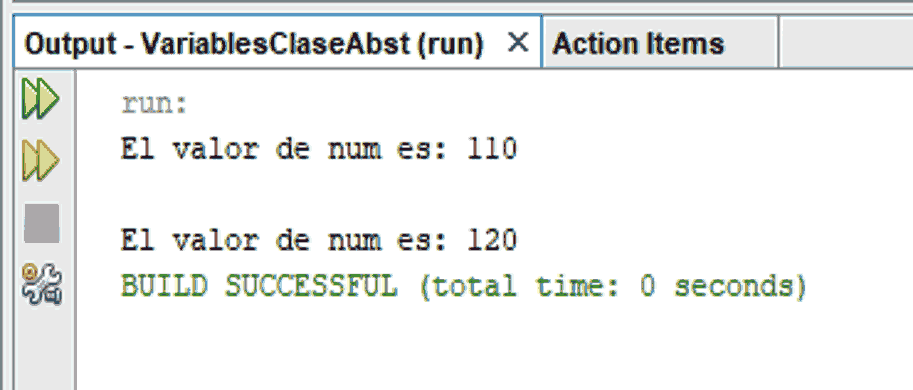
Como podemos ver en las llamadas al método calcula() se muestra el resultado que devuelve el **método implementado en la clase abstracta**, sin embargo en las llamadas al método calcula2() se devuelve el resultado del **método implementado en la clase del objeto que le invoca**.

## **Variables de Clase Abstractas y Polimorfismo.**

Ya dijimos en la página donde hablamos de [Clases Abstractas e Interfaces](https://codesitio.com/recursos-utiles-para-tu-web-o-blog/cursos/curso-de-java-clases-abstractas-e-interfaces/), que **no podemos crear objetos de ninguno de los dos**, pero si podemos crear **variables de Clase** y asignarlas referencias a **objetos de Clases que heredan o implementan dichas Clases Abstractas o Interfaces**. Veamos un ejemplo de ello, para ello tan solo mostraremos los cambios en la clase principal (método main) y la salida en consola, ya que el resto de clases son iguales:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | package variablesclaseabst;  public class VariablesClaseAbst {      /\*\*       \* @param args the command line arguments       \*/      public static void main(String[] args) {            //Creamos una Variable de Clase\_Abstrac          Clase\_Abstrac varAbst;            //Creamos objetos de Clase\_A y de Clase\_B          Clase\_A objA = new Clase\_A();          Clase\_B objB = new Clase\_B();            //Asignamos los objetos a la Variable de la clase abstracta          varAbst = objA;          System.out.println("El valor de num es: "+varAbst.calcula2());            System.out.println();            varAbst = objB;          System.out.println("El valor de num es: "+varAbst.calcula2());        }    } |

Y en consola tenemos la siguiente salida:



Esta forma de proceder usando variables de Clase y asignarlas después distintos objetos, simplifica el código. Empleamos mismas sentencias, solo cambia el objeto asignado. Si tenemos 2 o 3 objetos no hay mucha diferencia de usar un método u otro, el ahorro aparece cuando se incrementa el número de objetos.

Crear código de esta manera resulta más eficiente, con tiempos de ejecución menores y más fáciles de mantener. El ejemplo muestra una nueva faceta de**polimorfismo**