2. Tratamiento de objetos con JAVA

2.1 Ciclo de Vida de los Objetos

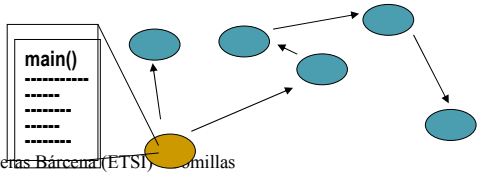
- Los objetos tienen un tiempo de vida y consumen recursos durante el mismo. Cuando un objeto no se va a utilizar más, debería liberar el espacio que ocupaba en la memoria de forma que las aplicaciones no la agoten (especialmente las grandes).
- La recolección y liberación de memoria es responsabilidad de un thread llamado Automatic Garbage Collector (recolector automático de basura).
- Este thread monitoriza a los objetos, marcando los que se han salido de alcance.

2.2 Secuencia de ejecución

- Todo programa java parte de una única clase clase.
- El código de esa clase que se ejecutará será el contenido de un método especial, llamado main, que será quien lance mensajes al resto de objetos.
- Toda clase puede contener el método main junto con la definición de atributos y métodos. (Esto no es lo más recomendable).
- En este curso diferenciaremos entre las clase ejecutables (poseen únicamente el método main) y las clases que definen objetos (definen atributos y métodos, pero nunca poseen main).

Nombre completo del método main:

```
public static void main(String s[])
```

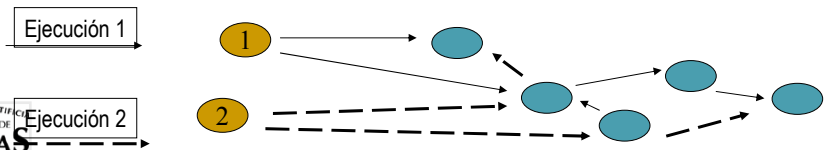


2.3 Clases – Estructura

- La estructura genérica de toda clase java es la siguiente:

```
class NombreClase
{
   [public static void main(String s[])
   { …..            }   ]
   [definición de atributos]
   [implementación de constructores]
   [definición/implementación de métodos]
}
```

- Se deberán realizar los programas separando las clases de ejecución de las clases que definen la lógica de los objetos:



2.3 Mensajes

- La sintaxis para lanzar un mensaje es la siguiente:

```
receptor.método(argumentos)
```

```
Persona p;
String s;
s = p.getNombre();
```

- También se puede hacer referencia directamente a los atributos de un objeto*:

```
s = p.nombre;
```

*Siempre y cuando los atributos formen parte del interfaz del objeto

2.3 Clases – Mi primer programa

- El mensaje que hace posible mostrar información por la salida estándar (pantalla) es: `System.out.println(String s);`

```
class HolaMundo
{
   public static void main(String s[])
   {
        System.out.println("Hola Mundo");
   }
}
```

2.4 Creación de un objeto

- Para la creación de una ocurrencia de una clase, hay que seguir los siguientes pasos:
  - Definir el tipo de objeto. Puntero a una zona de memoria.
  - Creación del objeto utilizando el operador new e invocando al constructor correspondiente (por defecto o personalizado). Reserva del espacio de memoria.

Constructor

El constructor es un método de clase que recibe su mismo nombre.

Su objetivo es crear e inicializar los atributos del objeto si se precisa.

Puede aceptar argumentos

2.4 Creación de un objeto (Ejemplo)

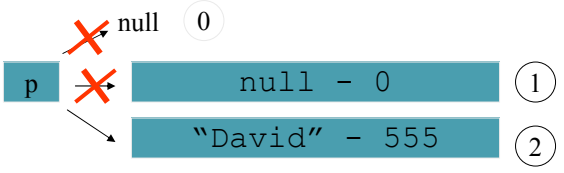
- Definir el tipo de objeto:

  (0) `Persona p;`

- Creación del objeto invocando al constructor correspondiente (por defecto o personalizado):

```
(1) p = new Persona();
(2) p = new Persona("David", 555);
```

P es una referencia a una zona de memoria donde se almacena una ocurrencia de la clase Persona (más información sobre referencias diap. 83)



2.4 Creación de un objeto (Ejemplo)

```
class Persona
{   int dni;
    String nombre;
}
```

```
class Persona
{   int dni;
    String nombre;
    Persona(int a, String b)
    {
        dni = a;
        nombre = b;
    }
}
```

No es necesario definirlo, es el constructor por defecto disponible para todas las clases.

Una vez que nos definimos un constructor personalizado para la clase, deja de estar disponible.


```
1.  p = new Persona();
2.  p = new Persona("David", 34343434);
```


2.4 Creación de un objeto (Ejemplo)

```
class Persona
{   int dni;
    String nombre;
}
```

```
class Persona
{   int dni;
    String nombre;
    Persona(int a, String b)
    {
        dni = a;
        nombre = b;
    }
    Persona()
    {
        dni=0; nombre=""; //P.e.
    }
}
```

No es necesario definirlo, es el Constructor por defecto disponible para todas las clases.

```
1.  p = new Persona();
2.  p = new Persona("David", 34343434);
```

2.4 Constructores

- En Java, en el momento que se define un constructor personalizado, el compilador supone que es el programador el que gestionará el resto de los constructores.
- Por este motivo, el constructor por defecto deja de estar disponible cuando se define alguno personalizado.
- Al definir un constructor en una subclase, se deberá tener en cuenta lo siguiente:
  - Para crear una ocurrencia de una subclase el compilador internamente creará primero una ocurrencia de la superclase y sobre ésta creará la definitiva.
  - Para crear la primera ocurrencia, invocará al constructor por defecto.
  - En el caso que no exista, se deberá indicar al compilador cuál debe utilizar en su lugar.
  - Para ello se utiliza la pseudovariable: **super**

2.5. Herencia – Ejemplo de Herencia y Clases Abstractas

- La forma de indicar al compilador que una clase hereda de otra es a través de la palabra reservada ***extends*.**

```
class Programador extends Persona
{
   ….
}
```

- Para definir una clase abstracta se deberá anteponer a *class* la palabra reservada ***abstract***. Lo mismo ocurrirá con los métodos abstractos.

```
public abstract class Dinosaurio
{
    protected byte masaCerebral;
    protected short peso;

    public abstract void darComida(int com);
    public abstract int getComido();
}
```

NOTA: Para más información acerca de herencia consultar el manual de POO. (pág 62)
Para más información acerca de clases abstractas consultar el manual de POO. (pág 71)

2.5 Constructores en la herencia - Error

```
class Persona
{   int dni;
    String nombre;
    Persona(int d, String n)
    {
        dni = d;
        nombre = n;
    }
}
```

Mensaje de error: Falta el constructor por defecto

```
new Trabajador(232, "aa", "EEE");
```

Este ejemplo generaría un error, ya que el compilador no podrá crear la ocurrencia inicial de Persona, puesto que no posee el constructor por defecto

```
class Trabajador extends Persona
{   String empresa;
    Trabajador(int d, String n, String e)
    {
        dni = d;
        nombre = n;
        empresa = e;
    }
}
```


2.5 Constructores en la herencia - Correcto

```
class Persona
{   int dni;
    String nombre;
    Persona(int d, String n)
    {
        dni = d;
        nombre = n;
    }
}
```

```
new Trabajador(232, "aa", "EEE");
```

Ahora, a través de super le estaremos indicando el constructor que debe ejecutar

```
class Trabajador extends Persona
{   String empresa;
    Trabajador(int d, String n, String e)
    {
        super(d, n);
        empresa = e;
    }
}
```

2.5 Interfaces - Definición

```
interface InterfaceA {
    métodoA( );
}
```

```
interface Interface1 {
    método1( );
    método2( );
}
```

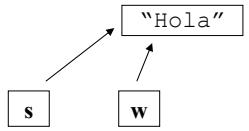
```
class Clase2 implements  Interface1,
InterfaceA
   {
   método1( )
   {
         implementación del método1
   }
   método2( )
   {
         implementación del método2
   }
   métodoA( )
   {
         implementación del métodoA
   }
 //métodos específicos
}
```

```
class Clase1 implements  Interface1
{
   método1( )
   {
         implementación del método1
   }
   método2( )
   {
         implementación del método2
   }
   //métodos específicos
}
```


2.6 Referencias

- Todas las variables son apuntadores o referencias a objetos.
- Cuando se realiza una asignación del tipo siguiente, se están creando dos referencias sobre el mismo objeto (alias). Cuando modifiquemos dichas referencias, se estará modificando el contenido del objeto:

```
String s = new String("Hola");
String w = s;
```



- En paso de parámetros se hace siempre uso de alias, es decir, se pasan los objetos por referencia. Los tipos de datos se pasan por valor.
- No existen los objetos locales, sólo referencias locales.
- Las referencias tienen ámbitos, los objetos no.

2.7 this y super

- this
  - Hace referencia a la ocurrencia en la que se está ejecutando un método.
  - Se utiliza para ejecutar métodos del mismo objeto.
  - Sustituye al receptor del mensaje cuando se trata del mismo objeto.
- super
  - Hace referencia a un método definido en la superclase.
  - Cuando nos encontramos en el constructor, hace referencia al de la clase padre.
  - En un método de instancia indica la ejecución de un método de la superclase con los datos del objeto en el que nos encontremos.

2.7 this y super - ejemplo

```
class Persona
{   int dni;
    String nombre;
    ....
    String a()
    {
        return "Mundo";
    }
}
```

```
class Trabajador extends Persona
{   String empresa;
    ...
    String a()
    {
        String s = this.b() + super.a();
        return s;
    }
    String b()
    {
        return "Hola ";
    }
}
```

```
Trabajador t = new Trabajador();
Persona p = new Persona();
p.a();  ──────────────►  "Mundo"
t.a();  ──────────────►  "Hola Mundo"
```

2.8 Reflectividad (Identificación de Tipos en Tiempo de Ejecución - RTTI) – java.lang.reflect

- El objetivo de la reflectividad es obtener información del objeto en tiempo de ejecución.
- Se puede saber de qué clase es una instancia, métodos o atributos que posee, etc.
- La clase principal a emplear es Class. Una ocurrencia de esta clase se obtiene ejecutando el método getClass() de la clase Object y nos proporcionará un objeto con toda la información de una clase.
- Relación de Class con otras clases del paquete java.lang.reflect
  - Field: Información de los atributos de la clase.
  - Method: Información de los métodos de la clase.
  - Constructor: Información de los constructores de la clase.

2.8 Reflectividad – Class (java.lang.reflect)

- Métodos:
  - Constructor[] **getDeclaredConstructors**()
  - Field[] **getDeclaredFields**()
  - Method[] **getMethods**()
  - boolean **isInstance** (Object obj)
    - Devuelve true si el objeto es del **tipo** de la clase sobre la que se ejecuta. Una ocurrencia de la clase Trabajador (Trabajador hereda de Persona), es del tipo Trabajador y Persona.
- Class no redefine el método equals. Por lo tanto, la clase a la que pertenece una ocurrencia será igual a otra, sólo cuando sean exactamente la misma. No se tendrá en cuenta la relación de herencia entre ellas.

```
Persona p=new Persona();
Class c=p.getClass();
boolean b=c.equals(Persona.class); //true
```

2.8 Reflectividad – instanceof

- Es un operador que devuelve un valor booleano.
- Se ejecuta sobre ocurrencias devolviendo si es de un determinado tipo o no.
- Sintaxis:

```
objeto instanceof nombreClase
```

  - `nombreClase: Persona, String, Coche`

- Ejemplo:

```
Persona p=new Persona();
boolean b=p instanceof Persona;
```

- Tiene el mismo funcionamiento que el método isInstance(Object).
- Se utilizará instanceof para un tratamiento único y estático para cada clase, mientras que isIntance(Object) para uno dinámico y aplicable a un conjunto de clases.

2.8 Reflectividad (Identificación de Tipos en Tiempo de Ejecución - RTTI) – java.lang.reflect - Ejemplo

```
class Clases
{
   public static void main(String s[]) throws Exception
   {
        Class clases[]=new Class[2];
        clases[0]=Persona.class; //Forma de hacer referencia a una
        clases[1]=Trabajador.class;  // clase

        Persona lista[]= new Persona[2];
        lista[0]=new Persona(111, "David");
        lista[1]=new Trabajador(222, "Manolo", 3000);

        System.out.println("Con instanceof");
        for(int i=0;i<lista.length;i++)
        {
          if(lista[i] instanceof Persona)
            System.out.println(lista[i] + " es del tipo Persona");
          if(lista[i] instanceof Trabajador)
            System.out.println(lista[i] + " es del tipo Trabajador");
```

2.8 Reflectividad (Identificación de Tipos en Tiempo de Ejecución - RTTI) – java.lang.reflect - Ejemplo

```
        //instanceOf sólo para comprobaciones individuales
        System.out.println("Con isInstance");
        for(int i=0;i<lista.length;i++)
        {
           for(int j=0;j<clases.length;j++)
           {
              if(clases[j].isInstance(lista[i]))
                System.out.println(lista[i] + " es del tipo " + clases[j]);
            }
        }
        System.out.println("Con Class");
        for(int i=0;i<lista.length;i++)
        {
           for(int j=0;j<clases.length;j++)
           {
              if(lista[i].getClass().equals(clases[j]))
                System.out.println(lista[i] + " es del tipo " + clases[j]);
           }
        }
    }
}
```

2.8 Reflectividad (Identificación de Tipos en Tiempo de Ejecución - RTTI) – java.lang.reflect - Ejemplo

---- Salida por pantalla ---

```
Con instanceof
Nombre: David - DNI: 111 es del tipo Persona
Nombre: Manolo - DNI: 222 - Sueldo: 3000 es del tipo Persona
Nombre: Manolo - DNI: 222 - Sueldo: 3000 es del tipo Trabajador
Con isInstance
Nombre: David - DNI: 111 es del tipo class Persona
Nombre: Manolo - DNI: 222 - Sueldo: 3000 es del tipo class Persona
Nombre: Manolo - DNI: 222 - Sueldo: 3000 es del tipo class
  Trabajador
Con Class
Nombre: David - DNI: 111 es del tipo class Persona
Nombre: Manolo - DNI: 222 - Sueldo: 3000 es del tipo class
  Trabajador
```