La Mediterrània

Repàs de conceptes

- ***Definició i Estructura d'una classe.***

```
import paquetAImportar.classeAImportar;  // importem una classe concreta
import paquetAImportar2.*;  // importem TOTES les classes del paquet paquetAImportar2

public class ClasseExemple  //Declaració d'una classe
{
  //Atributs de la classe
            modificador tipus nomAtribut_1;
            modificador tipus nomAtribut_2;
            ...
            modificador tipus nomAtribut_N;
  //Constructor de la classe
            public ClasseExemple(){
               //Solem inicialitzar els atributs de la classe en el constructor.
               // Si no els inicialitzem tots, almenys els que disposem d'informació en el
               // moment de crear l'objecte
            }
   // Mètodes de la classe
            modificador tipusDadaRetorn nomDelMètode_1( [paràmetres (opcional)]){
              //Instruccions que executarà el mètode
            }
            modificador tipusDadaRetorn nomDelMètode_2( [paràmetres (opcional)]){
              //Instruccions que executarà el mètode
            }
             ....
            modificador tipusDadaRetorn nomDelMètode_N( [paràmetres (opcional)]){
              //Instruccions que executarà el mètode
            }
   // Llistat dels getters i el setters
            // Per a l'atribut 1
            public void setNomAtribut_1(tipus v1){
                this.nomAtribut_1 = v1;
            }
            public tipus getNomAtribut_1(){
                return this.nomAtribut_1;
            }
            ...
            // Per a l'atribut N
            public void setNomAtribut_N(tipus vN){
                this.nomAtribut_N = vN;
            }
            public tipus getNomAtribut_N(){
                return this.nomAtribut_N;
            }
} // fi de la clase
```

- ***Encapsulació.***

Tracta sobre la manera d'ocultar com han estat implementat els atributs, d'un objecte. S'accedeix al atributs mitjançant els mètodes públics.

Per exemple, en una classe Vehicle, podem tindre un atribut anomenat ***velocitat***, que siga ***privat.*** L'única manera de poder modificar la velocitat seria utilitzant els mètodes accelerar() i frenar(), és a dir, aquestos mètodes encapsulen la velocitat. En la vida real, l'única manera que tenim de modificar la velocitat d'un vehicle és accelerant o frenant. Ens dóna igual que el vehicle siga una moto, un cotxe, una bici o un patinet. La manera en com es modifica la velocitat queda 'oculta' a la resta de les classes.

Generalment els mètodes d'accés als atributs s'anomenen getters i setters.

-*El Getter*:

És el mètode per accedir als atributs en forma de només lectura se'ls anomena "getters". Són els mètodes que retornen el valor dels atributs.

-El Setter:

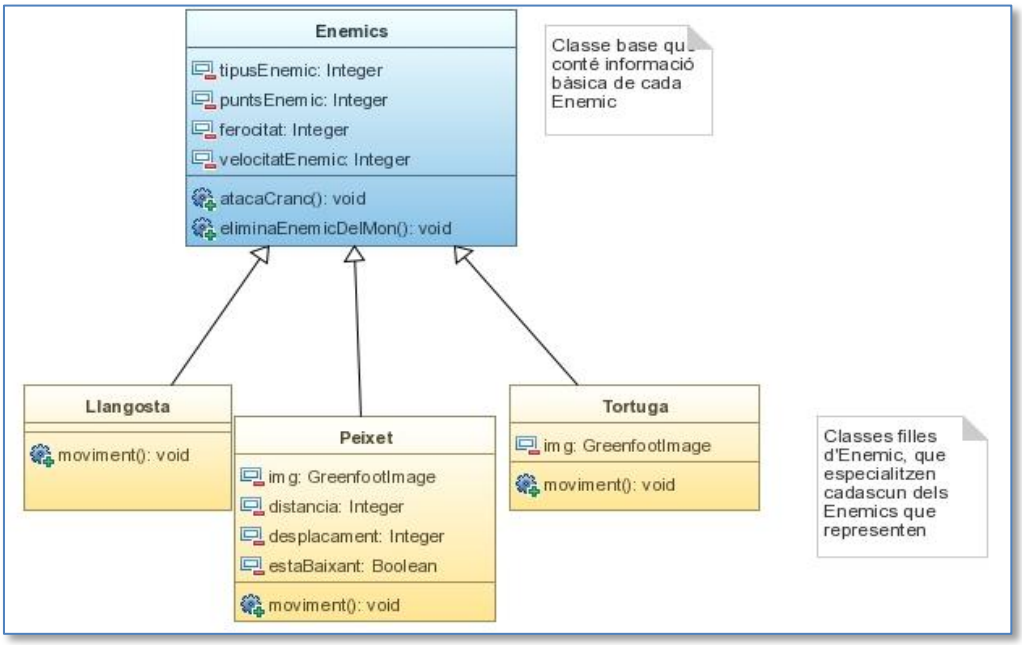
És el mètode per accedir als atributs en forma d'escriptura se'ls anomena "setters". Són els mètodes que estableixen el valor dels atributs.

Els *modificadors* que podem aplicar tant als atributs com als mètodes són els següents:

Mètode	Descripció
private	Un membre privat accessible només per a la classe en la qual està definit.
protected	La mateixa classe, les subclasses i totes les classes dins del mateix paquet tenen accés als membres qualificats amb aquest mètode d'accés.
public	Totes les classes tenen accés als membres públics de la classe.
package	Només les classes del mateix paquet que la classe tenen accés als membres

- ***Herència***

L'herència és la transmissió dels mètodes i atributs d'una classe a una altra. Gràcies a l'herència es poden establir jerarquies entre classes. Establir una jerarquia és un procés subjectiu, que depèn del programador i dels matisos d'apreciació de cadascú.



Hem vist com a la classe *Enemics* hem definit els atributs bàsics que tots els enemics han de tindre, així com algun mètode útil, que pot utilitzar qualsevol classe d'enemic (bé siga la Llangosta, el Peixet o la Tortuga).

Hem vist la necessitat de crear Constructors per a cadascuna de les *subclasses* per a poder **redefinir** adequadament **els atributs generals** de cada Enemic, segons el cas de cada classe **i inicialitzar els propis** atributs.

```
public Tortuga(){
    // Establim els atributs de la classe Enemic
    this.setFerocitat(30);
    this.setTipusEnemic(3);
    this.setVelocitatEnemic(6);
    this.setPuntsEnemic(30);
    //Posem la imatge de la tortuga reflexada
     horitzontalment (mirant cap a l'esquerre)
    img = new GreenfootImage("tortuga.png");
    img.mirrorHorizontally();
    setImage(img);

  }
```

Còdi Font 2 Constructor de la subclasse Tortuga

```
public Peixet(){
    // Establim els atributs de la classe Enemic
    this.setFerocitat(5);
    this.setTipusEnemic(2);
    this.setVelocitatEnemic(2);
    this.setPuntsEnemic(15);
    // Establim els atributs  propis de la classe Peixet
    this.distancia = 25;
    this.estaBaixant=true;
    //Generem un numero a l'atzar, per a vore quina de
     les tres imatges tindrà el peixet.
    int num = Greenfoot.getRandomNumber(3);
    switch(num){
      case 0: img = new GreenfootImage("peix1.png");
            break;
      case 1: img = new GreenfootImage("peix2.png");
            break;
      case 2: img = new GreenfootImage("peix3.png");
            break;
    }
```

Còdi Font 1Constructor de la subclasse Peixet

- ***Llenguatge Java***

A més a més hem utilitzat tipus primitius del llenguatge Java, com ***int***, ***boolean***.
Hem utilitzat condicionals ***if...else***, o ***switch ... case*** (amb el seu corresponent ***break***).
Hem declarat i inicialitzat variables.
Alguns poden haver utilitzat algun tipus de bucle (***for***, ***while***, ***do..while***).
Hem creat objectes de distintes classes (amb l'operador ***new***).
Hem cridat a mètodes (declarats per nosaltres, o d'altres que ens els proporcionaven les classes pare). Sempre amb el format

```
objecte.nomDelMètode();
Classe.nomDelMètode();
```

Hem utilitzat el modificador ***this***, quan ens volíem referir a l'objecte actual. Sempre que no l'utilitzarem dins d'un mètode ***static***.

Els atributs no cal inicialitzar-los (encara que és recomanable fer-ho), i les **variables/objectes** que **definim** dins d'un bloc (**entre la { i la }**) només **viuen dins d'eixe bloc**. Fora d'eixes {} no existeixen.

Hem aprés a reconèixer alguns errors que ens dóna el compilador, i a solucionar-los.
Hem utilitzar el mètode println() de la classe System.out per a mostrar per consola algun tipus d'informació que ens ajude a depurar el comportament del codi.

La Mediterrània

Repaso de conceptos

- ***Definición y Estructura de una clase.***

```
import paquetAImportar.claseAImportar;  // importamos una clase concreta
import paquetAImportar2.*;  // importamos TODAS las clases del paquete paquetAImportar2

public class ClaseExemple  //Declaración de una clase
{
  //Atributos de la clase
            modificador tipo nomAtribut_1;
            modificador tipo nomAtribut_2;
            ...
            modificador tipo nomAtribut_N;
  //Constructor de la clase
            public ClaseExemple(){
               //Solemos inicializar los atributos de la clase en el constructor.
               // Si no los inicializamos todos, al menos los que dispongamos de información en el
               // momento de crear el objeto
            }
   // Métodos de la clase
            modificador tipusDadaRetorn nomDelMétodo_1( [parámetros (opcional)]){
              //Instrucciones que ejecutará el método
            }
            modificador tipusDadaRetorn nomDelMétodo_2( [parámetros (opcional)]){
              //Instrucciones que ejecutará el método
            }
             ....
            modificador tipusDadaRetorn nomDelMétodo_N( [parámetros (opcional)]){
              //Instrucciones que ejecutará el método
            }
   // Listado de getters y setters
            // Per a l'atribut 1
            public void setNomAtribut_1(tipus v1){
                 this.nomAtribut_1 = v1;
            }
            public tipus getNomAtribut_1(){
                 return this.nomAtribut_1;
            }
            ...
            // Per a l'atribut N
            public void setNomAtribut_N(tipus vN){
                 this.nomAtribut_N = vN;
            }
            public tipus getNomAtribut_N(){
                 return this.nomAtribut_N;
            }
} // Fin de la clase
```

- ***Encapsulación.***

Trata sobre la manera de ocultar como han sido implementado los atributos de un objeto. Se accede al atributos mediante los métodos públicos.

Por ejemplo, en una clase **Vehículo**, podemos tener un atributo llamado **velocidad**, que sea privado. La única manera de poder modificar la velocidad sería utilizando los métodos **acelerar**() y **frenar**(), es decir, estos métodos encapsulan la velocidad. En la vida real, la única manera que tenemos de modificar la velocidad de un vehículo es acelerando o frenando. Nos da igual que el vehículo sea una moto, un coche, una bici o un patinete. La manera en cómo se modifica la velocidad queda oculta 'en el resto de las clases.

Generalmente los métodos de acceso a los atributos se denominan getters y setters.

-El Getter:

Es el método para acceder a los atributos en forma de sólo lectura se les llama "getters". Son los métodos que devuelven el valor de los atributos.

-El Setter:

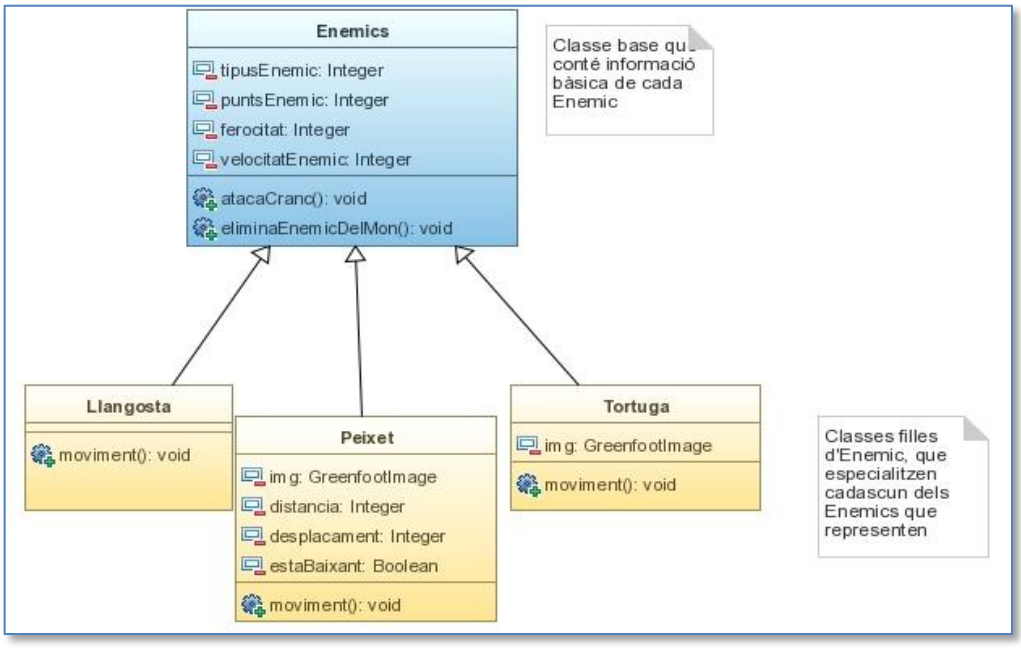
Es el método para acceder a los atributos en forma de escritura se les llama "setters". Son los métodos que establecen el valor de los atributos.

Los *modificadores* que podemos aplicar tanto a los atributos como a los métodos son los siguientes:

Método	Descripción
private	Un miembro privado accesible sólo para la clase en la que está definido.
protected	La misma clase, las subclases y todas las clases dentro del mismo paquete tienen acceso a los miembros calificados con este método de acceso.
public	Todas las clases tienen acceso a los miembros públicos de la clase.
package	Sólo las clases del mismo paquete que la clase tienen acceso a los miembros

- ***Herencia***

La herencia es la transmisión de los métodos y atributos de una clase a otra. Gracias a la herencia se pueden establecer jerarquías entre clases. Establecer una jerarquía es un proceso subjetivo, que depende del programador y los matices de apreciación de cada



uno.

Hemos visto como la clase Enemigos hemos definido los atributos básicos que todos los enemigos deben tener, así como algún método útil, que puede utilizar cualquier clase de enemigo (bien sea la langosta, el Pez o la Tortuga).

Hemos visto la necesidad de crear Constructores para cada una de las subclases para poder **redefinir** adecuadamente **los atributos generales** de cada Enemigo, según el caso de cada clase **e inicializar los propios** atributos.

```
public Tortuga(){
    // Establim els atributs de la classe Enemic
    this.setFerocitat(30);
    this.setTipusEnemic(3);
    this.setVelocitatEnemic(6);
    this.setPuntsEnemic(30);
    //Posem la imatge de la tortuga reflexada
     horitzontalment (mirant cap a l'esquerre)
    img = new GreenfootImage("tortuga.png");
    img.mirrorHorizontally();
    setImage(img);

  }
```

Código Fuente 4 Constructor de la subclase Tortuga

```
public Peixet(){
    // Establim els atributs de la classe Enemic
    this.setFerocitat(5);
    this.setTipusEnemic(2);
    this.setVelocitatEnemic(2);
    this.setPuntsEnemic(15);
    // Establim els atributs  propis de la classe Peixet
    this.distancia = 25;
    this.estaBaixant=true;
    //Generem un numero a l'atzar, per a vore quina de
     les tres imatges tindrà el peixet.
    int num = Greenfoot.getRandomNumber(3);
    switch(num){
      case 0: img = new GreenfootImage("peix1.png");
            break;
      case 1: img = new GreenfootImage("peix2.png");
            break;
      case 2: img = new GreenfootImage("peix3.png");
            break;
    }
```

Código Fuente 3Constructor de la subclase Peixet

- ***Lenguaje Java***

Además hemos utilizado tipos primitivos del lenguaje Java, como ***int***, ***boolean***.
Hemos utilizado condicionales ***if...else***, o ***switch ... case*** (con su correspondiente ***break***).
Hemos declarado e inicializado variables.
Algunos pueden haber utilizado algún tipo de bucle (***for***, ***while***, ***do..while***).
Hemos creado objetos de distintas clases (con el operador ***new***).
Hemos llamado a métodos (declarados por nosotros, o de otros que nos les proporcionaban las clases padre). Siempre con el formato

objeto.nombreDelMétodo();
Clase.nombreDelMétodo();

Hemos utilizado el modificador ***this***, cuando nos queríamos referir al objeto actual. Siempre que no se utilizan dentro de un método ***static***.
Los atributos no hace falta inicializarlos (aunque es recomendable hacerlo), y las **variables/objetos que definimos** dentro de un bloque (**entre la {y}**) sólo **viven dentro de ese bloque**. Fuera de esas {} no existen.

Hemos aprendido a reconocer algunos errores que nos da el compilador, y solucionarlos.
Hemos utilizar el método println () de la clase System.out para mostrar por consola algún tipo de información que nos ayude a depurar el comportamiento del código.