Programació

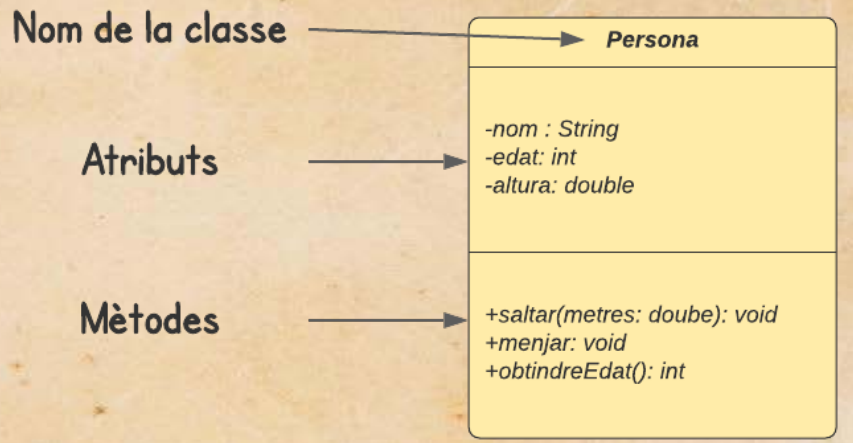
7.2 POO en Java

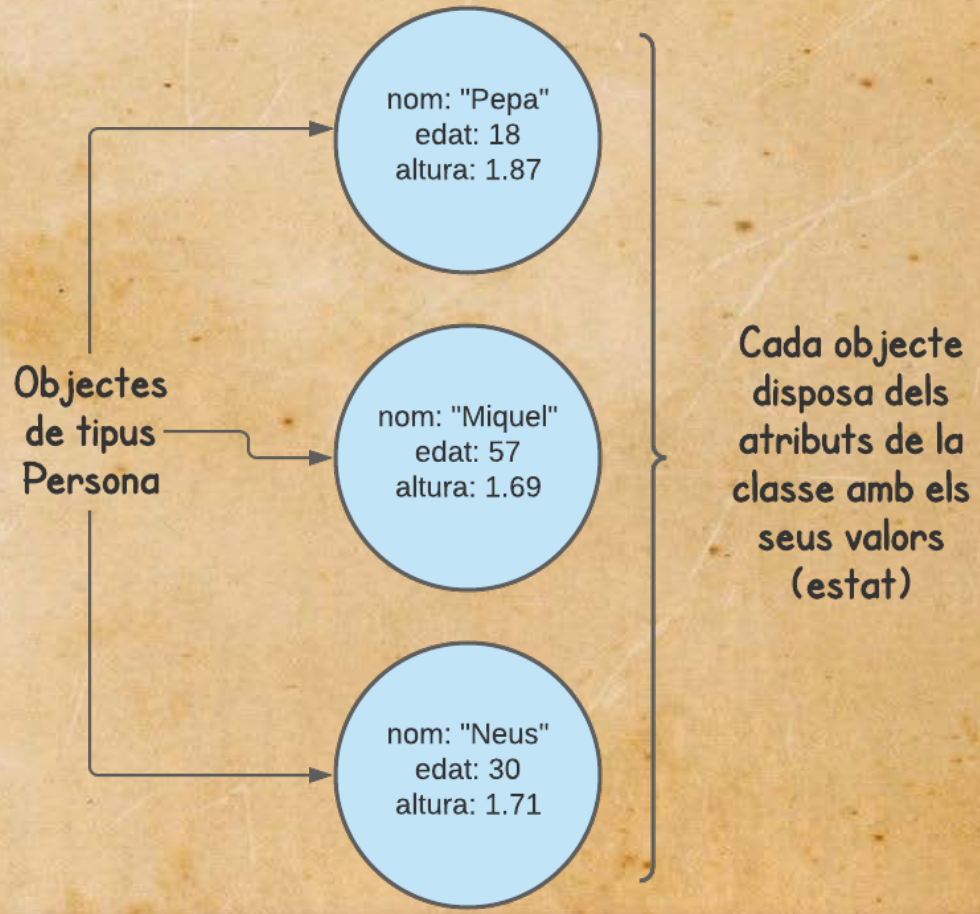
Definint classes i objectes

Una classe en Java utilitza variables per definir els atributs i mètodes per definir les operacions.

Addicionalment, existeixen uns mètodes especials coneguts com a constructors que són invocats normalment per a crear i **inicialitzar un objecte que acabem de crear**.

Definint classes i objectes





Referències en Java

- El **comportament** dels objectes en la memòria de l'ordinador i les seues operacions elementals (creació, assignació i destrucció) **és idèntic al dels arrays**. Això és pel fet que tant objectes com arrays utilitzen referències.

- Abans de construir un objecte cal **declarar una variable** el tipus de la qual siga la seua classe. La diferència és que mentre que la variable de tipus primitiu emmagatzema directament un valor, la de tipus referencia emmagatzema **la referència d'un objecte**.

```
Persona p; //p es una variable de tipus persona (referència)
```

Operador *new*

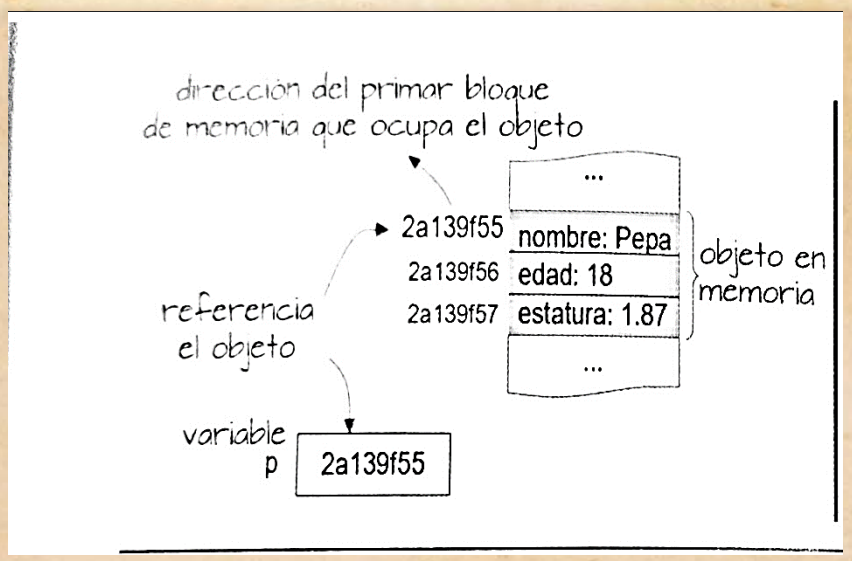
- La manera de **crear objectes**, com per als arrays, és mitjançant l'operador ***new***.

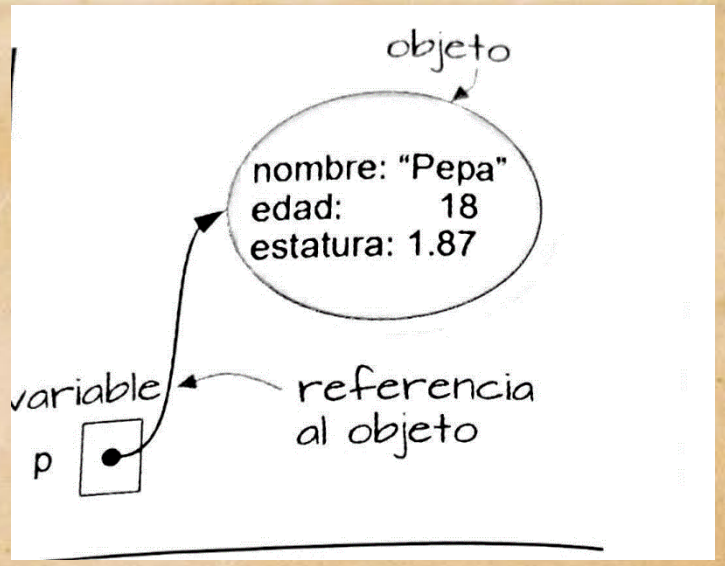
```
p = new Persona();
```

- En este cas es **crea un objecte** de tipus persona, i **s'assigna la seua referència a la variable p**. L'operador ***new*** busca en memòria un buit disponible on construir l'objecte, ocupa els blocs de memòria que necessite i finalment retorna la referència de l'objecte recentment creat, que s'assigna a la variable p

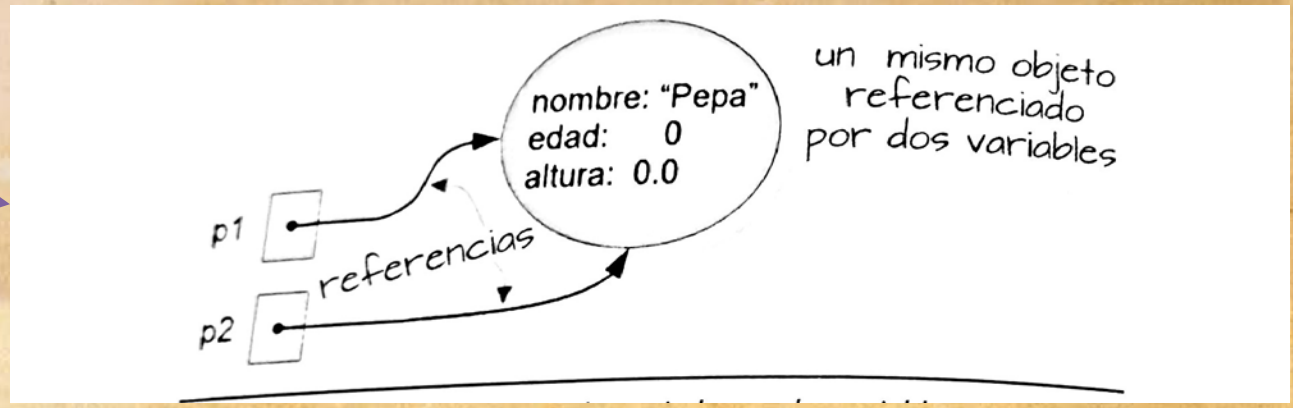
Qué imprimiria System.out.println(p) ?

Operador *new*






```
Persona p1, p2;
p1 = new Persona();
p2 = p1;
p2.nombre = "Pepa"
```



Referència a *null*

- El valor literal “null” és una referència nul·la. Dit d’altra manera, una referència a cap bloc de memòria.
- Quan declarem una variable referència s’inicialitza per defecte a null.
- Si intentem accedir als membres d’una referència nul·la es produirà un error i acabarà l’execució del programa:

```
Persona p; // S’inicialitza per defecte a null
p.obtindreEdat(); // Error de tipus Null Pointer Exception
```

És ùtil quan volem que una variable deixe de fer referència a un objecte:

```
Persona p = new Persona(); // p referencia a un objecte
p = null; // p no referencia res.
```

Recol·lector de memòria brossa.

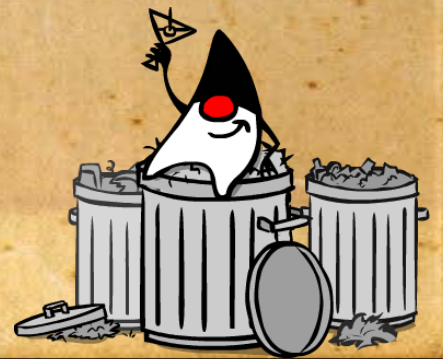
Hi ha tres formes d'aconseguir que un objecte no estiga referenciat:

- Crear un objecte i no assignar-lo a cap variable.
  ```
  new Persona();
  ```
- Assignar a null una variable que tenia una referencia a objecte.
  ```
  Persona p = new Persona();
  p = null;
  ```
- Assignar un objecte distint a una variable.
  ```
  Persona p = new Persona();
  p = new Persona();
  ```

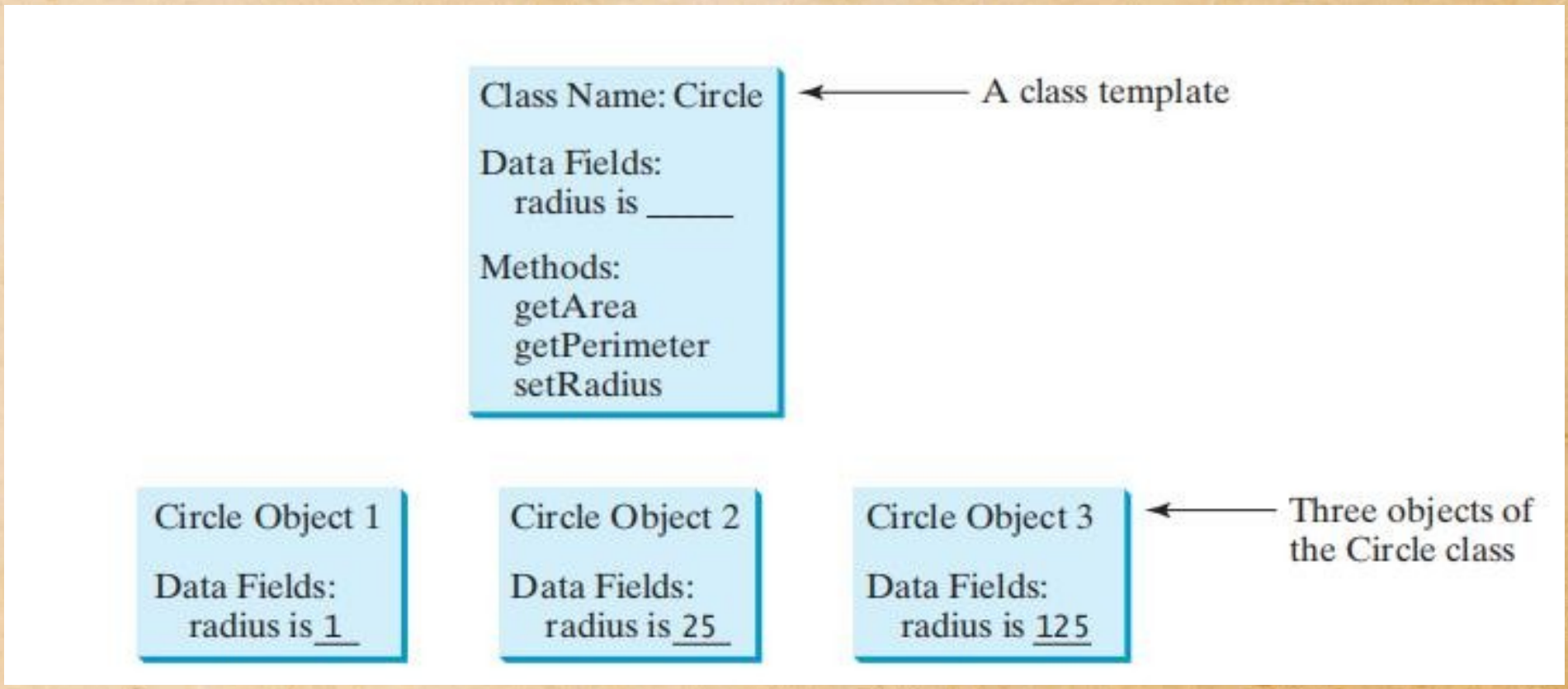
Recol·lector de memòria brossa.

- En tots eixos casos l'objecte **queda perdut en memòria**, es a dir, no hi ha forma de poder accedir a ells i es queden consumint memòria.

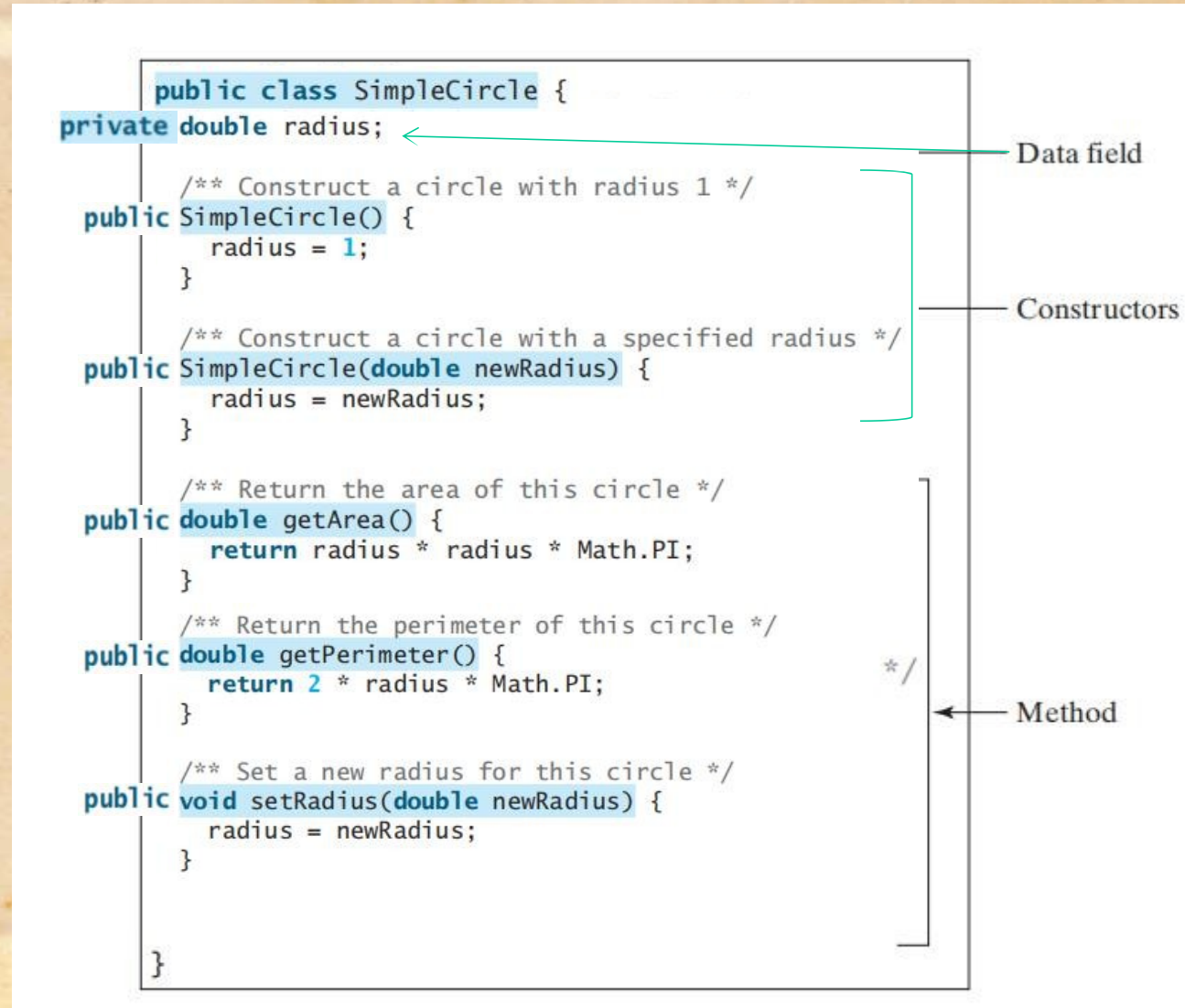
- Per a evitar este problema, Java disposa d'un mecanisme anomenat "**recol·lector de brossa**" (garbage collector) que s'executa periòdicament de forma transparent a l'usuari i va **destruint els objectes que no estiguen referenciats**, alliberant la memòria que ocupen.



Exemple de classe i objectes:
Circle



Exemple de definició de una classe



Creació d'objectes de classes pròpies

Tal i com hem vingut fent amb les classes del API de Java, podem crear objectes de classes definides per nosaltres mateixos.

Com hem dit abans, fent ús de l'operador **`new`** es crea un objecte invocant al **constructor** i s'inicialitzen les seues dades (li donem un estat a l'objecte).

Creació d'objetes per a provar-los

```
public static void main(String[] args) {
  // Create a circle with radius 1
  SimpleCircle circle1 = new SimpleCircle();
  System.out.println("The area of the circle of radius "
    + circle1.radius + " is " + circle1.getArea());

  // Create a circle with radius 25
  SimpleCircle circle2 = new SimpleCircle(25);
  System.out.println("The area of the circle of radius "
    + circle2.radius + " is " + circle2.getArea());

  // Create a circle with radius 125
  SimpleCircle circle3 = new SimpleCircle(125);
  System.out.println("The area of the circle of radius "
    + circle3.radius + " is " + circle3.getArea());

  // Modify circle radius
  circle2.radius = 100;
  System.out.println("The area of the circle of radius "
    + circle2.radius + " is " + circle2.getArea());
}
```

Pràctica 1

Escriu la clase `SimpleCircle` en un fitxer `Java` y crea en el paquet de test del projecte el mètode main `TestSimpleCircle` que contindrà el main plantejat a la diapositiva anterior.

Prova el seu funcionament.

Creació de mètodes d'una clase

```
public <tipus1> <nomMètode>(<tipus2> <parametre>, ...)
```

on el **tipus1** indica el tipus del valor retornat, que pot ser:

- void (res)
- <tipus/Classe> (un valor de tipus primitiu o una referència a un objecte)
- <tipus/Classe>[] (una referencia a un vector de valors de tipus primitiu o de referències a objectes)
- <tipus/Classe>[][] (una referència a una matriu de valors de tipus primitiu o de referències a objectes) …

    - on <nomMètode> començarà en minúscula
    - on el tipus2 pot ser qualsevol tipus (excepte void)
    - on tots els paràmetres (0 a N) són passats per valor

Un altre exemple

Suposem que necessitem crear una classe que tinga les característiques d'un televisor per a un programa que necessitarà d'eixe tipus de objectes.

Després d'un anàlisi previ arribem a la conclusió de que les propietats d'un televisor que ens interessen i identifiquen al televisor al nostre problema són:

- **el canal que s'està veient**
- **el nivell de volum**
- **encesa o apagada**

Un altre exemple

Les operacions que pot realitzar el televisor són:

- **encendre**
- **apagar**
- **pujar el volum**
- **baixar el volum**
- **canviar al canal següent**
- **canviar al canal anterior**

Pràctica 2

Seguint l'estructura de la classe SimpleCircle, defineix la classe `TV` i crea un fitxer `TestTV.java` per provar el seu funcionament.

Crea **dos televisors**, realitza direferents operacions amb ells i comprova que l'estat després d'eixes operacions és l'esperat.

Sobrecàrrega de mètodes

- Funcionalitat que permet tindre **mètodes diferents amb el mateix nom.**
- La llista d'arguments (paràmetres d'entrada) deu ser diferent per **identificar inequívocament** a cada mètode que es diga igual.
- No es pot sobrecarregar un mètode en funció del seu paràmetre d'eixida.

Sobrecàrrega de mètodes. Exemple

- Donada una classe `Persona` es poden definir estos dos mètodes:

```
public void caminar(int passos) {
    //Aquí implementaria el mètode passos(int)
}
public void caminar(double passos) {
    //Aquí implementaria el mètode passos(double)
}
```

```
public int caminar(int numPassos) {
    //ERROR
}
```

Construint objectes i usant constructors

Els constructors **són un tipus especial de mètodes** que tenen TRES particularitats:

- Han de tindre **el mateix nom que la classe**
- No tenen **tipus de dada de retorn** (ni void)
- Els constructors **els invoquem usant l'operador new** (que és el que crea l'objecte) quan volem inicialitzar-lo.

Construint objectes i usant constructors

Altres missions dels constructors poden ser:

- Recollir els valors per tenir-los en compte a l'hora de crear l'objecte.
- Gestionar els errors que puguen aparèixer en la fase d'inicialització.
- Aplicar processos, més o menys complicats, en les quals pot intervenir tot tipus de sentències (condicionals o repetitives).

Si el constructor fa més d'una tasca (complexe) és convenient modularitzar-lo.

Declaració de constructors.

- Una mateixa classe pot disposar de **diversos constructors**. La diferència entre ells la determina l'ordre i tipus dels paràmetres d'entrada que li proporcionem
- Si no es defineix cap constructor s'autodefineix un **constructor per defecte** (que serà un constructor buit, sense paràmetres).
- Si es defineix algun constructor, **no es proporcionarà el constructor buit per defecte** (es tindria que definir expressament).

Exemple de un possible constructor de la classe Persona

```
public Persona (String unDni,
  String unNom, int unaEdat) {
   dni = unDni;
   nom = unNom;
   if (unaEdat >= 0 && unaEdat <= 150)
      edat = unaEdat;
}
.....
  // Després es definiran els
     seus mètodes
```

Exemple de TestPersona.java

```
public static void main(String args[]) {
   Persona p1 = new Persona("00000000","Pepe Gotera",33);
   Persona p2 = new Persona(); //Aquí hi haurà un error
   System.out.println("Visualització de persona p1:");
   p1.visualitzar();
   System.out.println(" Visualització de persona p2:");
   p2.visualitzar(); // Imaginem que hem creat el constructor buit.
}
```

Exemple: Classe Persona (v1.0)

```
Visualització de persona p1:
Dni...........:00000000
Nom...........:Pepe Gotera
Edat..........:33
Visualización de persona p2:
Dni...........:null
Nom...........:null
Edat..........:0
```

Pràctica 3

- Crea un nou projecte que contindrà la classe `Persona` i defineix el mètode `visualitzar` que mostrarà per pantalla les dades d'una persona.
- Crea la classe `TestPersona` on escriuràs el mètode main de la diapositiva anterior. Apareixerà un error en una de les línies del mètode main. Soluciona'l.
- Visualitza les dades d'ambdues persones i raona per què apareixen els resultats que apareixen.

Exercici “CompteCorrent” (I)

Dissenya la classe *CompteCorrent* que emmagatzeme les dades: DNI i nom del titular, així com el saldo. Les operacions típiques amb un compte corrent són:

- **Crear un compte**: Es necessita el DNI i nom del titular. El saldo inicial serà 0.0
- **Traure diners:** el mètode ha d'indicar si ha sigut possible dur a terme l'operació (si existeix saldo suficient)
- **Ingressar diners:** S'incrementa el saldo.
- **Mostrar informació:** Mostra la informació disponible del compte corrent (nom, DNI i saldo del compte)

Pregunta

Què estic fent en cada línia?

```
1. int x;
2. Interval r;
3. Interval r = new Interval();
4. int[] t;
5. t = new int[100];
6. Interval[] tt;
7. Interval[] tt = new Interval[100];

8.    for (int i = 0; i < tt.length; i++) {
        tt[i] = new Interval();
    }
```

Referenciació a objectes

```
// Declaració de la variable   de referència obj1 no inicialitzada (valdrà null)
X obj1;

// Creació de l'objecte al que es podrà accedir via la variable de referència obj1

obj1 = new X(...);

// Declaració de la variable   de referència obj2 i creació de l'objecte al que es podrà
accedir via la variable de referència obj2

X obj2 = new X(...);

// Declaració de la variable de referència obj3 no inicialitzada (valdrà null)


X obj3;

// La variable obj3 fa referència al mateix objecte que fa referència la variable obj1

obj3 = obj1;

// Declaració de variable de referència que fa referència al mateix objecte que fa
// referència la variable obj2

X obj4 = obj2;
```

Valor null i NullPointerException

Recorda que les variables de referència no inicialitzades per defecte tenen el valor null.

Accedir a una propietat (amb el operador .) de una variable null provocarà que aparega l'excepció

NullPointerException

```
public class ShowErrors {
  public void method1() {
    Circle c;
    System.out.println("What is radius "
      + c.getRadius());
    c = new Circle();
  }
}
```

Pregunta

Per què falla este programa?

```
class Test {
  public static void main(String[] args) {
    A a = new A();
    a.print();
  }
}

class A {
  String s;

  public A(String newS) {
    s = newS;
  }

  public void print() {
    System.out.print(s);
  }
}
```

Definició de atributs

- Si t'has donat compte, els **atributs d'una classe** seran el que fins ara coneixíem com a variables globals, no deixant de ser globals a la classe per este fet.
- És molt important definir els modificadors correctament per a mantenir els principis d'abstracció i encapsulació.

Modificadors d‘accés o de visibilitat

- Una classe serà visible per altra, o no, depenent si es troben en el mateix **paquet** i dels **modificadors d’accés** que utilitze.

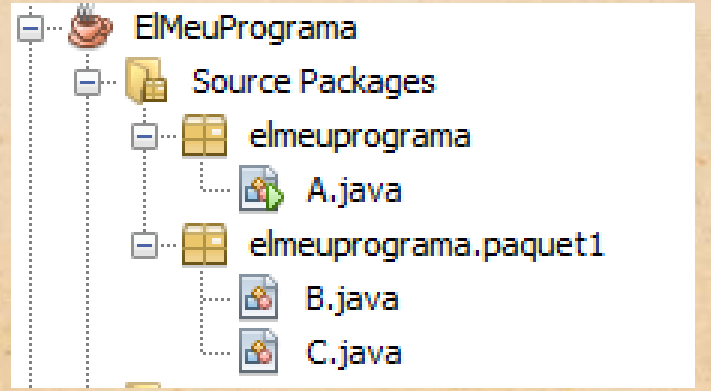
- Estos modificadors **canvien la seua visibilitat** entre classes, permetent que es mostre o s’oculte.

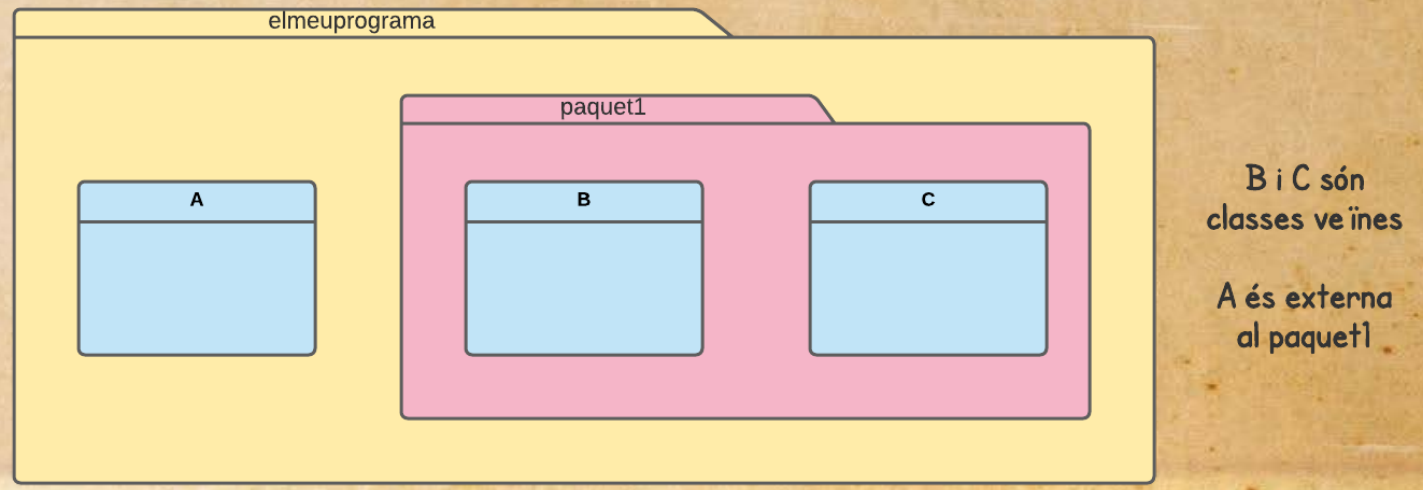
- Igual com podem modificar la visibilitat **entre classes**, també es pot modificar la visibilitat **entre membres** de distintes classes, es a dir, quins atributs i quins mètodes son visibles per a altres classes.

Modificadors d'accés per a classes

Degut a l'estructura de classes de Java, organitzades en paquets, dos classes qualsevol poden definir-se com a:

- **Classes veïnes**: Quan ambdues pertanyen al mateix paquet.
- **Classes externes**: Quan s'han definit en paquets distints.





B i C són classes veïnes

A és externa al paquet1

Modificadors d'accés per a classes

Podrem trobar dos tipus de visibilitats per a les classes:

- **Visibilitat per defecte**: Quan definim una classe sense utilitzar cap modificador d'accés. Eixa classe **només** serà visible per a les seues **classes veïnes**.
- **Visibilitat total:** Quan definim una classe utilitzant el modificador d'accés "**public**". Eixa classe serà **visible per a qualsevol altra classe**, siga veïna o no. Si la classe és externa caldrà que **importe** la classe a la que vol tindre accés.

```
package elmeuprograma.paquet1;

class B { // Sense modificador d'accés.
    /*
        Segons la nostra estructura, B
        serà visible per a C (veïna) pero
        B no serà visible per a A (externa)
    */
}
```

```
package elmeuprograma.paquet1;

public class B { // Modificador public.

}
```

```
package elmeuprograma;
import elmeuprograma.paquet1.B;
class A {
    // ...
```

Modificadors d'accés per a classes

	Visible des de ...	
	Classes veïnes	Classes externes
Sense modificador	✓	X
public	✓	✓

Modificadors d'accés per a membres

- D'igual manera que és possible modificar la visibilitat d'una classe, podem regular la visibilitat dels seus membres.
- Que un atribut siga visible significa que podem accedir a ell, tant per a llegir com per a modificar-lo. Que un mètode siga visible significa que pot ser invocat.
- Perquè un membre siga visible és indispensable que la seua classe també ho siga.
- Qualsevol membre és sempre visible dins de la seua pròpia classe, independentment del modificador d'accés que tinga.

```
package elmeuprograma;

public class A { // Classe pública
    int dada; // El seu àmbit és tota la classe.
    // L'atribut 'dada' és accessible des de qualsevol lloc d'A
}
```

Modificadors d‘accés per a membres

Podrem trobar els següents tipus de visibilitat per a membres:

- **Per defecte (default o package):** Quan definim un membre sense especificar cap modificador d’accés. Eixos membres seran visibles només des de les classes veïnes.
- **Pública (public):** Quan definim un membre amb el modificador d’accés “public”. Serà visible per a qualsevol altra classe, siga o no veïna.
- **Privada (private):** Quan definim un membre amb el modificador d’accés “private”. Serà visible només per a la pròpia classe que els defineix (ni veïnes ni externes).
- **Protegida (protected):** Quan definim un membre amb el modificador d’accés “protected”. Eixos membres seran visibles per a les seues classes veïnes i per a les seues subclasses encara que siguen externes.

Modificadors d'accés per a membres

	Visible des de ...		
	La pròpia classe	Classes veïnes	Classes externes
private	✓	X	X
sense modificador	✓	✓	X
protected	✓	✓	Només classes derivades o subclasses
public	✓	✓	✓

Exercici “CompteCorrent” (II)

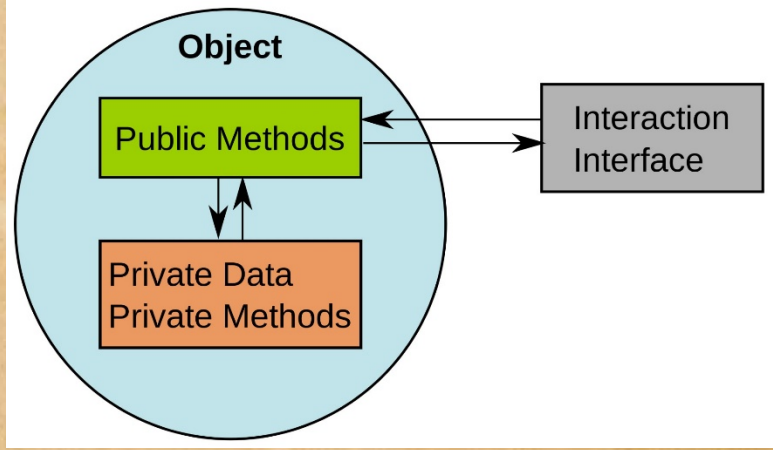
Modifica la visibilitat de la classe “CompteCorrent” per tal que siga visible des de les classes externes i la visibilitat dels seus membres siga:

- **saldo**: no serà visible per a altres classes.
- **nom**: serà visible per a qualsevol altra classe.
- **dni**: només serà visible per a les classes veïnes.
- Els constructors i altres mètodes seran visibles per a qualsevol classe.

Mètodes *get* i *set*

Un atribut públic pot ser llegit i modificat des de qualsevol classe el que planteja greus inconvenients:

- Eliminem el **principi d'ocultació**, deixant al descobert els detalls d'implementació (no ens interessa que siga visible tota l'estructura de l'objecte, només una certa part de la nostra encapsulació).
- - No podem **controlar els valors assignats** a estos atributs, que poden tindre o no sentit. Per exemple es pot assignar a l'edat un valor negatiu.



Mètodes *get* i *set*

Per eixe motiu existeix una **convenció** entre programadors que consisteix a **ocultar els atributs**, i en el seu lloc crear dos mètodes públics per a cadascun dels atributs als quals vulguem donar visibilitat (hi haurà atributs amb tots dos, només un, o cap dels mètodes get/set).

- El primer mètode, **set**, permetrà assignar un valor a un atribut, podent validar o manipular este valor abans de ser assignat a l'atribut.
- El segon, **get**, retorna el valor de l'atribut o bé alguna transformació del mateix (o còpia defensiva)

Mètodes *get* i *set*

Per convenció estos mètodes s'identifiquen amb set/get seguit del nom de l'atribut. Per exemple per a l'atribut "edat":

```
public class Persona {
    private int edat;

    public int getEdat() {
        return edat;
    }

    public void setEdat(int edat) {
        if(edat >=0){
            this.edat = edat;
        }
    }
```

Classe Circle amb atributs privats

```java
public class CircleWithPrivateDataFields {
  /** The radius of the circle */
  private double radius = 1;

  /** The number of objects created */
  private static int numberOfObjects = 0;

  /** Construct a circle with radius 1 */
  public CircleWithPrivateDataFields() {
    numberOfObjects++;
  }

  /** Construct a circle with a specified radius */
  public CircleWithPrivateDataFields(double newRadius) {
    radius = newRadius;
    numberOfObjects++;
  }

  /** Return radius */
  public double getRadius() {
    return radius;
  }

  /** Set a new radius */
  public void setRadius(double newRadius) {
    radius = (newRadius >= 0) ? newRadius : 0;
  }

  /** Return numberOfObjects */
  public static int getNumberOfObjects() {
    return numberOfObjects;
  }

  /** Return the area of this circle */
  public double getArea() {
    return radius * radius * Math.PI;
  }
}
```

Pas de objectes com paràmetres

- És possible passar objectes com a paràmetres en els mètodes plantejats.

```
public class Test {
  public static void main(String[] args) {
    // CircleWithPrivateDataFields is defined in Listing 9.8
    CircleWithPrivateDataFields myCircle = new
      CircleWithPrivateDataFields(5.0);
    printCircle(myCircle);
  }

  public static void printCircle(CircleWithPrivateDataFields c) {
    System.out.println("The area of the circle of radius "
      + c.getRadius() + " is " + c.getArea());
  }
}
```

Exercici “CompteCorrent” (III)

Modifica la classe per a ocultar tots els seus atributs. Després has d’implementar els mètodes necessaris per a poder llegir els valors dels atributs “saldo”, “nom” i “dni” i per a poder modificar els atributs “nom” i “dni” (assegura’t que el DNI continga 9 lletres)

Pregunta

Quina és l'eixida d'este codi?

```
public class Test {
  public static void main(String[] args) {
    Count myCount = new Count();
    int times = 0;

    for (int i = 0; i < 100; i++)
      increment(myCount, times);

    System.out.println("count is " + myCount.count);
    System.out.println("times is " + times);
  }

  public static void increment(Count c, int times) {
    c.count++;
    times++;
  }
}
```

```
public class Count {
  public int count;

  public Count(int c) {
    count = c;
  }

  public Count() {
    count = 1;
  }
}
```

Pregunta

Quina és l'eixida d'este codi?

```
public class Test {
  public static void main(String[] args) {
    Circle circle1 = new Circle(1);
    Circle circle2 = new Circle(2);

    swap1(circle1, circle2);
    System.out.println("After swap1: circle1 = " +
      circle1.radius + " circle2 = " + circle2.radius);

    swap2(circle1, circle2);
    System.out.println("After swap2: circle1 = " +
      circle1.radius + " circle2 = " + circle2.radius);
  }

  public static void swap1(Circle x, Circle y) {
    Circle temp = x;
    x = y;
    y = temp;
  }

  public static void swap2(Circle x, Circle y) {
    double temp = x.radius;
    x.radius = y.radius;
    y.radius = temp;
  }
}

class Circle {
  double radius;

  Circle(double newRadius) {
    radius = newRadius;
  }
}
```

Pregunta

Què mostra el següent codi?

```
public class Test {
  public static void main(String[] args) {
    int[] a = {1, 2};
    swap(a[0], a[1]);
    System.out.println("a[0] = " + a[0]
      + " a[1] = " + a[1]);
  }

  public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }
}
```

(a)

Pregunta

Què mostra el següent codi?

```
public class Test {
  public static void main(String[] args) {
    int[] a = {1, 2};
    swap(a);
    System.out.println("a[0] = " + a[0]
      + " a[1] = " + a[1]);
  }

  public static void swap(int[] a) {
    int temp = a[0];
    a[0] = a[1];
    a[1] = temp;
  }
}
```

(b)

Pregunta

Què mostra el següent codi?

```
public class Test {
  public static void main(String[] args) {
    T t = new T();
    swap(t);
    System.out.println("e1 = " + t.e1
      + " e2 = " + t.e2);
  }

  public static void swap(T t) {
    int temp = t.e1;
    t.e1 = t.e2;
    t.e2 = temp;
  }
}

class T {
  int e1 = 1;
  int e2 = 2;
}
```

(c)

Pregunta

Què mostra el següent codi?

```
public class Test {
  public static void main(String[] args) {
    T t1 = new T();
    T t2 = new T();
    System.out.println("t1's i = " +
      t1.i + " and j = " + t1.j);
    System.out.println("t2's i = " +
      t2.i + " and j = " + t2.j);
  }
}

class T {
  static int i = 0;
  int j = 0;

  T() {
    i++;
    j = 1;
  }
}
```

(d)

Pregunta

Què mostra el següent codi?

```java
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = null;
    m1(date);
    System.out.println(date);
  }

  public static void m1(Date date) {
    date = new Date();
  }
}
```

(a)

Pregunta

Què mostra el següent codi?

```java
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date = new Date(7654321);
  }
}
```

(b)

Pregunta

Què mostra el següent codi?

```java
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date.setTime(7654321);
  }
}
```

(c)

Pregunta

Què mostra el següent codi?

```java
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date = null;
  }
}
```

(d)

Ocultació d'atributs

- Sabem que dos variables declarades en àmbits niuats no poden tindre el mateix identificador ja que això genera un error.

```
public static void main(String[] args){
    int valor = 3;
    for(int i = 0; i < 100; i++){
        double valor = i;
    }
```

- Tanmateix, existeix una excepció quan una variable local en un mètode té el mateix identificador que un atribut de la classe. En este cas, dins del mètode, la variable local té prioritat sobre l'atribut. En l'argot de la POO es diu que la variable local oculta a l'atribut

```
public class Persona {
    int edat; // atribut enter

    void metode(){
        double edat; // variable local
        edat = 8.2;  // oculta l'atribut "edat"
        System.out.println(edat);
    }
```

La paraula reservada `this`

- Especialment útil per tractar la manipulació de atributs.
- Té dos finalitats:
  - **Dins dels mètodes no estàtics:** per fer referència l'objecte sobre el qual s'executa el mètode (És recomanable sempre utilitzar-lo per a evitar conflictes)

  Per exemple: this.edat o this.nom.

  - **Dins del mètode constructor:** per a invocar a un altre constructor de la mateixa classe (constructor sobrecarregat).

  Per exemple: this(edat, nom) invocaria al constructor Persona(int edat, String nom));

Exemple

- Creem un constructor de Persona que se crearà a partir de una altra (constructor còpia):

```
public Persona (Persona p) {
        this(p.dni, p.nom, p.edat);
}
```

Exemple

- Suposem que creem un mètode que crea una persona a partir d'una altra anomenat clonar:

```
public Persona clonar() {
    return new Persona(this);
}
```

Exemple

```
public static void main(String args[]) {
   Persona p1 = new Persona("00000000","Pepe Gotera",33);
   Persona p2 = new Persona(p1);
   Persona p3 = p1.clonar();
   System.out.println("Visualització de persona p2:");
   p2.visualitzar();
   System.out.println("Visualització de persona p3:");
   p3.visualitzar();
}
```

p1, p2 i p3 fan referència al mateix objecte o a objectes distints?

Pràctica 4

- Modifica la classe `Persona`, anteposant l'accés de qualsevol dels seus atributs, la paraula reservada `this`.

Pràctica 5

- Amplia el mètode main que tens en este moment, per a que es cree una persona a partir d'una altra, fent ús del mètode "clonar" i comprova que funciona correctament mostrant els valors de la nova persona creada.
- Crea una nova persona fent ús directament del “constructor còpia” y mostra els atributs de la persona creada.

Práctica 6

- Crea un nou constructor en Persona, que crearà una nova persona a partir d'altres dos. En este cas, el dni i el nom de la nova persona creada serán els de la persona1 i l'edat la de la persona 2.

Práctica 7

- Per últim, crea una nova versió del mètode “clonar” que utilitzarà el constructor creat anteriorment. En este cas, la persona1 que necessita este constructor, serà l’objecte que efectua l’operació y la persona2 se rebrà com a paràmetre.
- Invoca este mètode des del mètode main i comprova que funciona correctament.

Exercici “CompteCorrent” (IV)

Sobrecarrega la classe CompteCorrent per a poder crear objectes amb:

- El DNI, el nom del titular, i el saldo inicial.
- El DNI del titular i un saldo inicial.

Escriu un programa que comprove el funcionament dels mètodes.

(*) Utilitza també “this” per accedir als atributs d’instància.

Exercici “CompteCorrent” (V)

Existeixen gestors que administren els comptes bancaris i atenen els seus propietaris. Cada compte, té un únic gestor (o cap). Dissenya la classe **Gestor** de la qual ens interessa guardar el seu nom, telèfon, i l'import màxim autoritzat amb el qual pot operar. Respecte als gestors existeixen les següents restriccions:

- Un gestor tindrà sempre un nom i un telèfon.
- Si no s'assigna, l'import màxim autoritzat per operació serà de 10000 euros.
- Un gestor, després de ser creat no podrà canviar el seu número de telèfon i tothom podrà consultar-lo.
- El nom del gestor serà públic i l'import màxim només serà visible per a classes veïnes.

Modifica la classe CompteCorrent perquè puga disposar d'un objecte Gestor. Escriu els mètodes necessaris.