GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 2:

Programació Orientada a Objectes

(4) solució exercicis

Laura Igual
Departament de Matemàtica Aplicada i Anàlisi
Facultat de Matemàtiques
Universitat de Barcelona

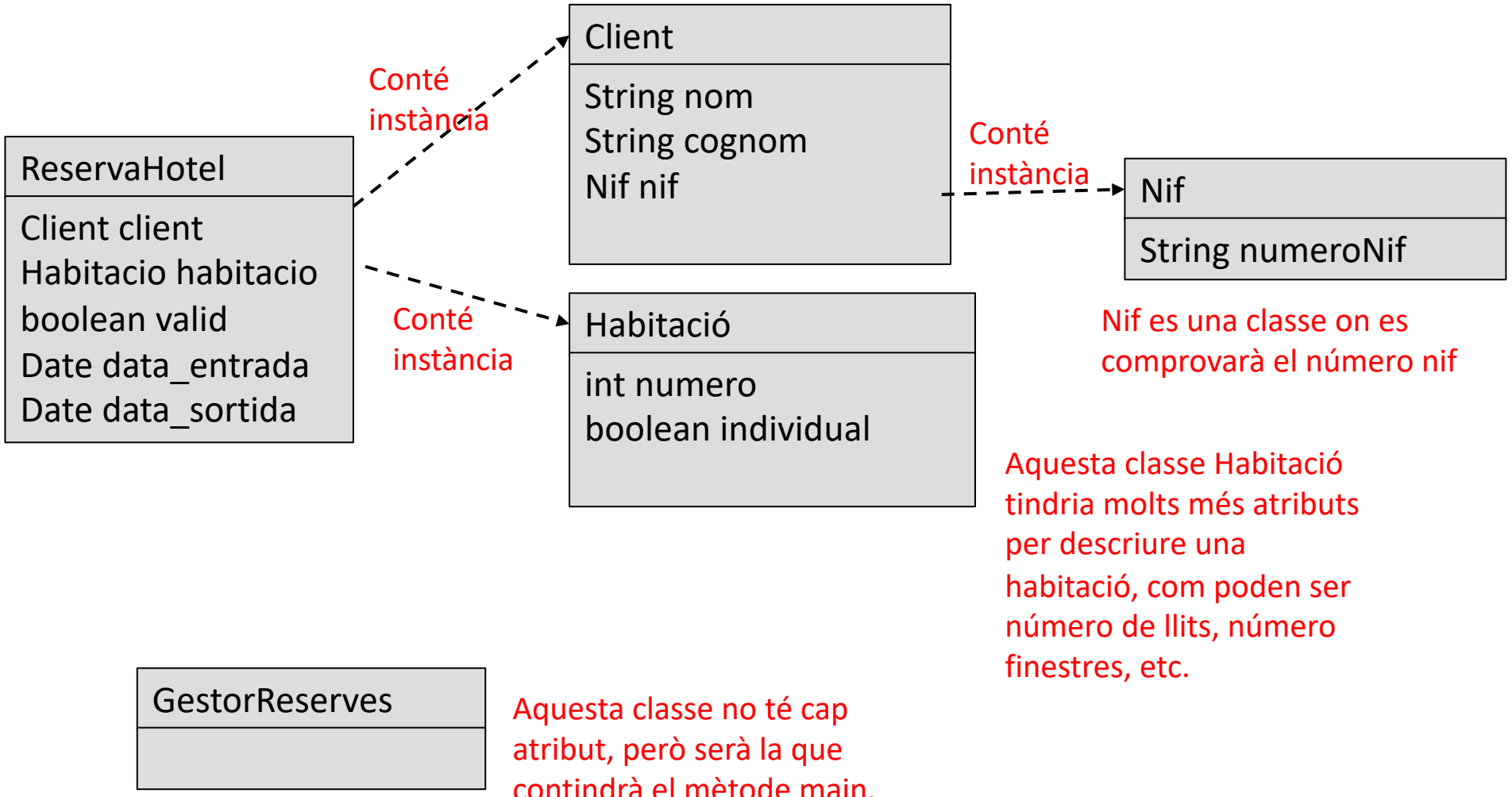
SOLUCIÓ EXERCICIS PLANTEJATS

Exercici 1

- Es vol implementar una aplicació de gestió de reserves d'hotel seguint el diagrama de classes definit a continuació. On apareixen el nom de les classes i la llista dels seus atributs.
- Implementa les classes:
  - **ReservaHotel**
  - **Habitacio**
  - **Client**
  - **GestorReserves**

- Al mètode main de la classe **GestorReserves** s'ha de crear una reserva i validar-la.

Exercici 1: Diagrama de classes



Solució Implementació

```
public class GestorReserves {
   public static void main(String[] args){
      Habitacio habitacio = new Habitacio(1);
      Client client = new Client("0000000E");
       // Demanar la informació sobre el client
       // Crear una nova reserva del hotel:
      ReservaHotel novaReserva = new ReservaHotel(habitacio, client);
      // Validar quan ja s'ha pagat la reserva:
      novaReserva.validar();
   }
}
```

No cal definir el constructor per defecte si no vols crear una reseva sense assignar habitació al client.

El constructor per defecte existeix mentre no es sobrecarregui amb qualsevol conjunt de parametres (inclos sense parametres).

```
public class ReservaHotel{
  Habitacio habitacio;
  Client client;
  Date data_entrada;
  Date data_sortida;
  boolean valid;
  // constructor de la classe:
  public ReservaHotel(Habitacio habitacio, Client client){
        this.habitacio = habitacio;
        this.client = client;
        valid = false;
  }
  // mètode per validar la reserva:
  public void validar(){
     valid = true;
  }
// més mètodes ...
}
```

```
public class Habitacio{
    int numero;
    boolean individual;
    // constructor de la classe:
    public Habitacio(int numero){
       this.numero= numero;
       this. individual = false;
     }
    public Habitacio(int numero, boolean individual){
         this.numero= numero;
         this. individual = individual;
     }
    //… setters & getters …
}
```

Solució Implementació

Una altra opció d'Implementació de la classe ReservaHotel:

```
public class ReservaHotel{
  Habitacio habitacio;
  Client client;
  Date data_entrada;
  Date data_sortida;
  boolean valid = false;
  // constructors de la classe:
  public ReservaHotel(){
        valid = false;
  }
  public ReservaHotel(Habitacio habitacio, Client client){
        this();
        this.habitacio = habitacio;
        this.client = client;
  }
  // mètode per validar la reserva:
  public void validar(){
     valid = true;
  }
// més mètodes ...
}
```

Si volem crear una reseva sense assignar habitació i client ho podem fer així.

Solució Implementació

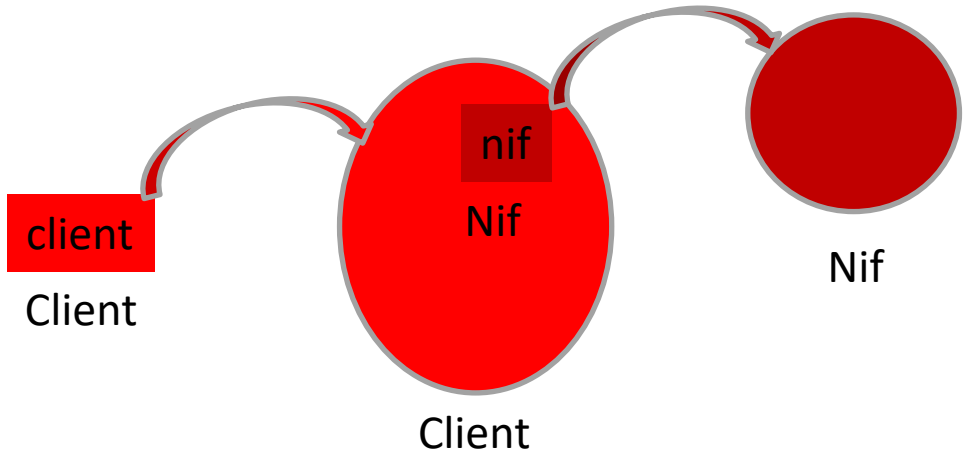
```
public class Client{
   String nom;
   String cognom;
   Nif nif;

// constructor de la classe:
   public Client(String nif){
      nif = new Nif(nif);
   }
    // mètodes d'accés I d'escriptura de la classe:
   public void setNom(String nom){
      this.nom=nom;
   }
   public void setCognom(String cognom){
      this.cognom = cognom;
   }
    public String getNom(){
      return this.nom;
   }
   public String getCognom(){
      return this.cognom;
   }
  // més mètodes
}
```

Sempre que creem un nou client ha de tenir un Nif associat.

Observació

```
public class GestorReserves {
   public static void main(String[] args){
       Habitacio habitacio = new Habitacio(1);
       Client client = new Client("44444444P");
        // demanar la informació sobre el client
        // Crear una nova reserva del hotel:
       ReservaHotel novaReserva = new ReservaHotel(habitacio, client);
       // Validar quan ja s'ha pagat la reserva:
       novaReserva.validar();
   }
}
```

Quan instanciem un objecte de la classe Client, estem instanciant un objecte de la classe Nif.



Exercici 2

Donada la classe Cotxe, implementeu les següents classes:

1. **CotxeAmbGPS** que hereta de **Cotxe** i tindrà dos atributs nous de tipus double: latitud i longitud i un nou mètode canviarCoordenades que rebrà dos doubles com a paràmetres.

2. **Taxi** que hereta de **Cotxe** i redefinirà el mètode recorrer de manera que imprimeixi per pantalla dos missatges, abans (“iniciar carrera”) i després (“fin de carrera”) de fer el recorregut.

Exercici 2

3. **FactoriaDeCotxes** que tindrà un mètode fabricarCotxeNou que retorni un cotxe de nova creació que algunes vegades serà un **Cotxe** un **CotxeAmbGPS** o un **Taxi**.

```
package ub.estudiant;
public class Cotxe{
	private String propietari;
	private String matricula;
	private double compteKilometres;

	public void vendre(String elPropietari) {
		propietari = elPropietari;
	}

	public void matricular(String laMatricula) {
		matricula = laMatricula;
	}

	public void recorrer(double kms){
		 compteKilometres = compteKilometres + kms;
	}

	public void printInfo(){
		String tmp = "Propietari: " + propietari + "; " +
		" Matricula: " + matricula + "; " +
		"Kms recorridos: " + compteKilometres  + ";" ;
		System.out.println(tmp) ;
	}
}
```

```
package ub.estudiant;
public class CotxeAmbGPS extends Cotxe{
	private double latitut = 0;
	private double longitut = 0;


	public void canviarCoordenades(double deltaLatitut, double deltaLongitut) {
		latitut = latitut + deltaLatitut;
		longitut = longitut + deltaLongitut;
	}

	public double getLatitut(){
		return latitut;
	}

	public double getLongitut(){
		return longitut;
	}

	public void printInfoPosicio(){
		String tmp  = "Latitut: " + latitut + ";" +  " Longitut: " + longitut + ";";
		System.out.println(tmp) ;
	}
}
```

```
package ub.estudiant;
public class DemoCotxeAmbGPS {
	public static void main(String[] args){
		 CotxeAmbGPS cotxe = new CotxeAmbGPS();

		// mètodes en CotxeAmbGPS
		cotxe.canviarCoordenades(0.01, 0.02);
		cotxe. printInfoPosicio();

		// mètodes en Cotxe, que es criden igual.
		cotxe.vendre("X.X.X");
		cotxe.matricular("PMM-000");

		cotxe.printInfo();
	}
}
```

```
package ub.estudiant;


public class Taxi extends Cotxe{

	public void recorrer(double kms){
		System.out.println("Taxi@: inicia carrera");
		printInfo();
		super.recorrer(kms);
		System.out.println("Taxi@: fin carrera");
	}

}
```

```
package ub.estudiant;
public class FactoriaDeCotxes{
	private int n=0;

	public Cotxe fabricarCotxeNou() {
		Cotxe elCotxe = null;
		if (n==0) {
		    elCotxe = new Cotxe();
		} else if(n==1){
		    elCotxe = new CotxeAmbGPS();
		} else{
		    elCotxe = new Taxi();
		}
		n = (n+1) % 3;
		return elCotxe;
	}

}
```

```
package ub.estudiant;
public class ExemplePolimorfisme{

    public static void main(String[] args){
            FactoriaDeCotxes factoria = new FactoriaDeCotxes();
            for(int i=0; i<10; i++){
                Cotxe cotxe = factoria.fabricarCotxeNou();
                // cotxe pot ser de diferents tipus.


                cotxe.vendre("X.X.X" + i);
                cotxe.matricular("PMM-" + i);

                cotxe.recorrer(i);

                cotxe.printInfo();
            }
    }
}
```

Quin mètode s'executa aquí?

Quin mètode s'executa aquí?

Com es pot deduir d'aquest codi, la repetició d'una mateixa instrucció pot donar lloc a l'execució de diferents peces de codi en cada ocasió.

Només en el moment mateix d'executar aquesta instrucció es decideix quin és el mètode triat: lligadura dinámica!

Font exercici 2

- https://ocw.ehu.eus/file.php/116/intro_java/mod-ii/notas/tema-7x.pdf