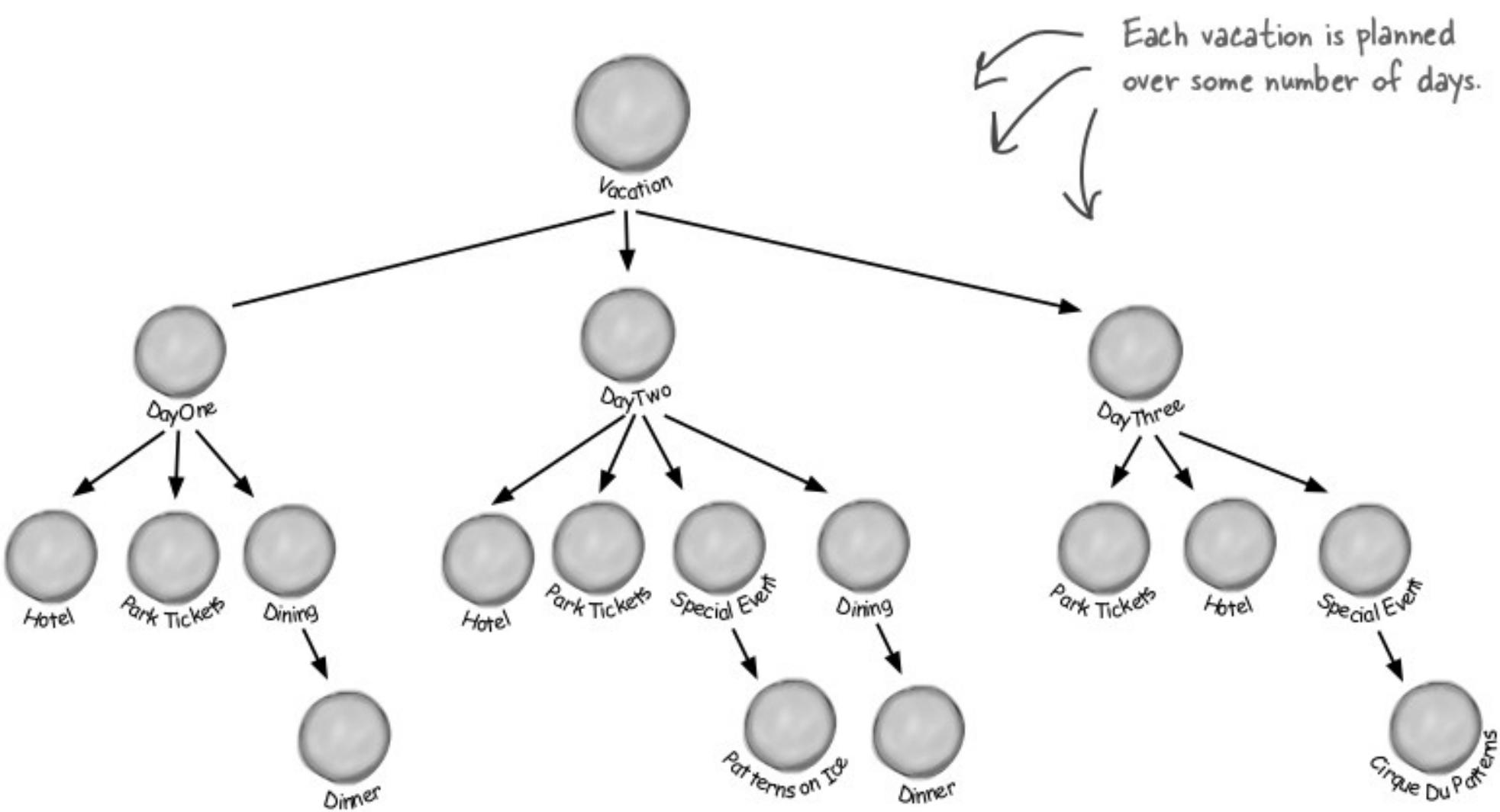
Builder

Nueva solicitud

- Te acaban de pedir que construyas un planificador de vacaciones para Patternsland, un nuevo parque temático en las afueras de Objectville. Los huéspedes del parque pueden elegir un hotel y varios tipos de entradas, hacer reservas en restaurantes e incluso reservar eventos especiales. Para crear un planificador de vacaciones, debe ser capaz de crear estructuras como esta:


Each vacation is planned over some number of days.
Vacation
DayOne
DayTwo
DayThree
Hotel
Park Tickets
Dining
Dinner
Hotel
Park Tickets
Special Event
Dining
Patterns on Ice
Dinner
Park Tickets
Hotel
Special Event
Cirque Du Patterns
Each day can have any combination of hotel reservations, tickets, meals and special events.

Necesitas un diseño flexible

- Cada huesped puede variar en la cantidad de días y tipos de actividades que incluye.
- Por ejemplo, un residente local puede no necesitar un hotel, pero quiere hacer reservas para cenas y eventos especiales. Otro huésped podría volar a Objectville y necesita un hotel, reservas para cenar y boletos de admisión.

- Por lo tanto, necesita una estructura de datos flexible que pueda representar todos los planes de los huespedes y todas sus variaciones;

- también necesita seguir una secuencia de pasos potencialmente complejos para crear el planificador.

¿Cómo puede proporcionar una forma de crear una estructura compleja sin mezclar con los pasos para crearla?

Builder

- Debes usar el patrón Builder para encapsular la construcción de un producto y permitir que se construya en pasos.

¿Por qué usar el patrón de construcción?

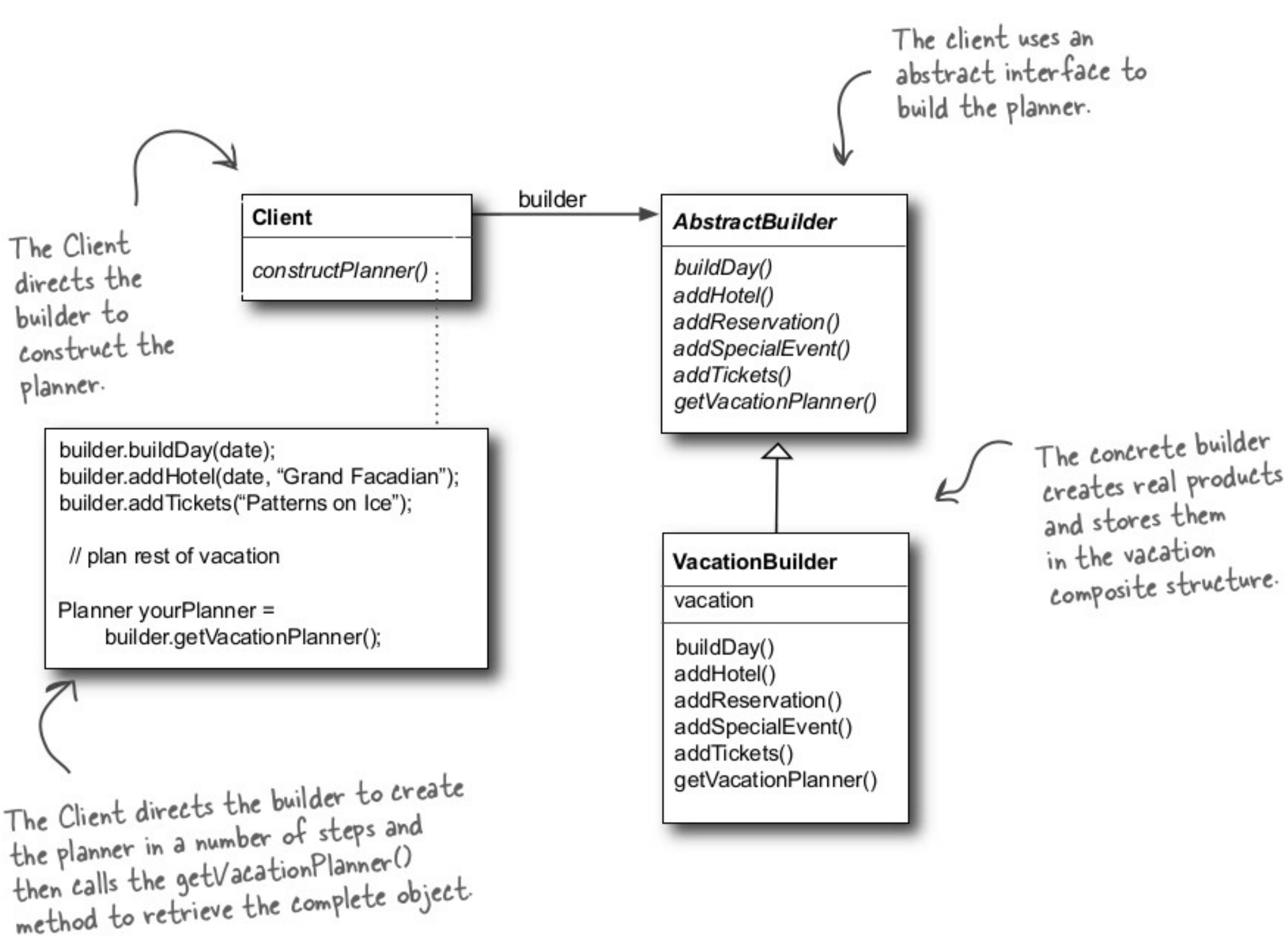
- ¿Recuerdas Iterator? Encapsulamos la iteración en un objeto separado y ocultábamos la representación interna de la colección del cliente.

¿Por qué usar el patrón de construcción?

- Es la misma idea aquí: encapsulamos la creación del planificador de viajes en un objeto (llamémoslo un “builder”(constructor)) y le solicitamos al builder que construya la estructura del planificador de viajes.

- El cliente ordena al constructor que construya el planificador.
- El cliente usa una interfaz abstracta para construir el planificador.

- El constructor concreto crea productos reales y los almacena en la estructura compuesta de vacaciones
- El cliente ordena al constructor que cree el planificador en una serie de pasos y luego llama al método getVacationPlanner() para recuperar el objeto completo.


The client uses an abstract interface to build the planner.
Client
constructPlanner()
builder
AbstractBuilder
buildDay()
addHotel()
addReservation()
addSpecialEvent()
addTickets()
getVacationPlanner()
The Client directs the builder to construct the planner.
builder.buildDay(date);
builder.addHotel(date, "Grand Facadian");
builder.addTickets("Patterns on Ice");
// plan rest of vacation
Planner yourPlanner =
builder.getVacationPlanner();
The concrete builder creates real products and stores them in the vacation composite structure.
VacationBuilder
vacation
buildDay()
addHotel()
addReservation()
addSpecialEvent()
addTickets()
getVacationPlanner()
The Client directs the builder to create the planner in a number of steps and then calls the getVacationPlanner() method to retrieve the complete object.

Beneficios del constructor

- Encapsula la forma en que se construye un objeto complejo.
- Permite que los objetos se construyan en un proceso de varios pasos y variable (a diferencia de las fábricas que es un solo paso).

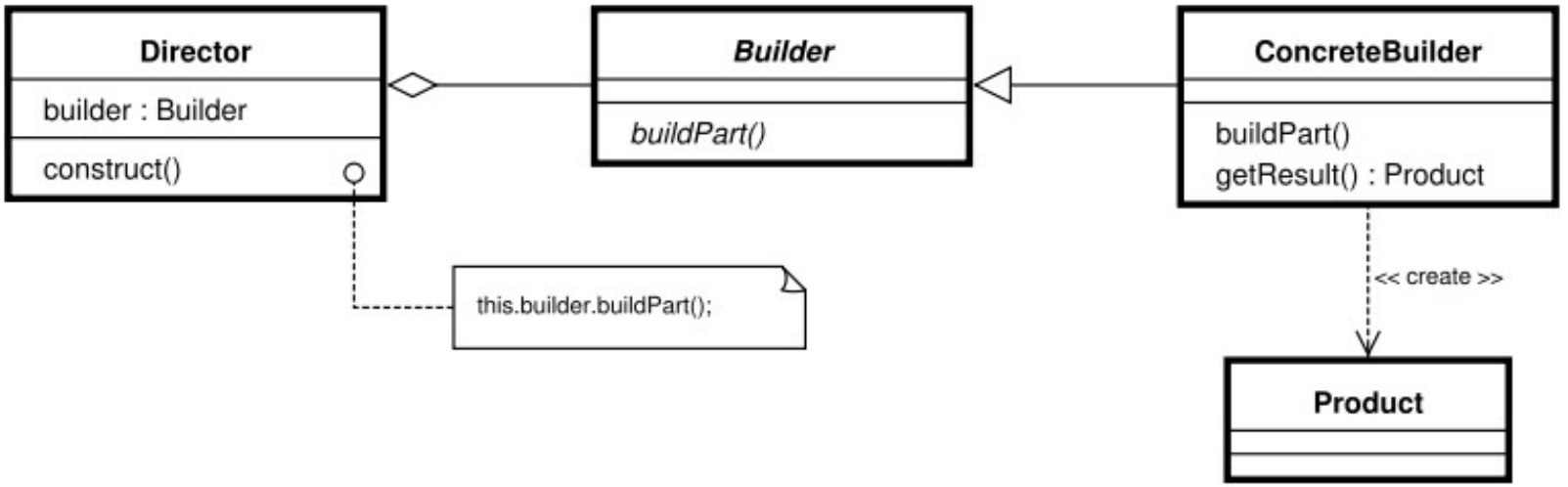
Beneficios del constructor

- Oculta la representación interna del producto del cliente.
- Las implementaciones de productos se pueden intercambiar porque el cliente solo ve una interfaz abstracta.

Usos e inconvenientes del constructor

- A menudo se usa para construir estructuras compuestas.
- La construcción de objetos requiere más conocimiento del dominio del cliente que cuando se utiliza el patron de Factory.

- Builder
  - interfaz abstracta para crear productos.
- Concrete Builder
  - implementación del Builder
  - construye y reúne las partes necesarias para construir los productos
- Director
  - construye un objeto usando el patrón Builder
- Producto
  - El objeto complejo bajo construcción



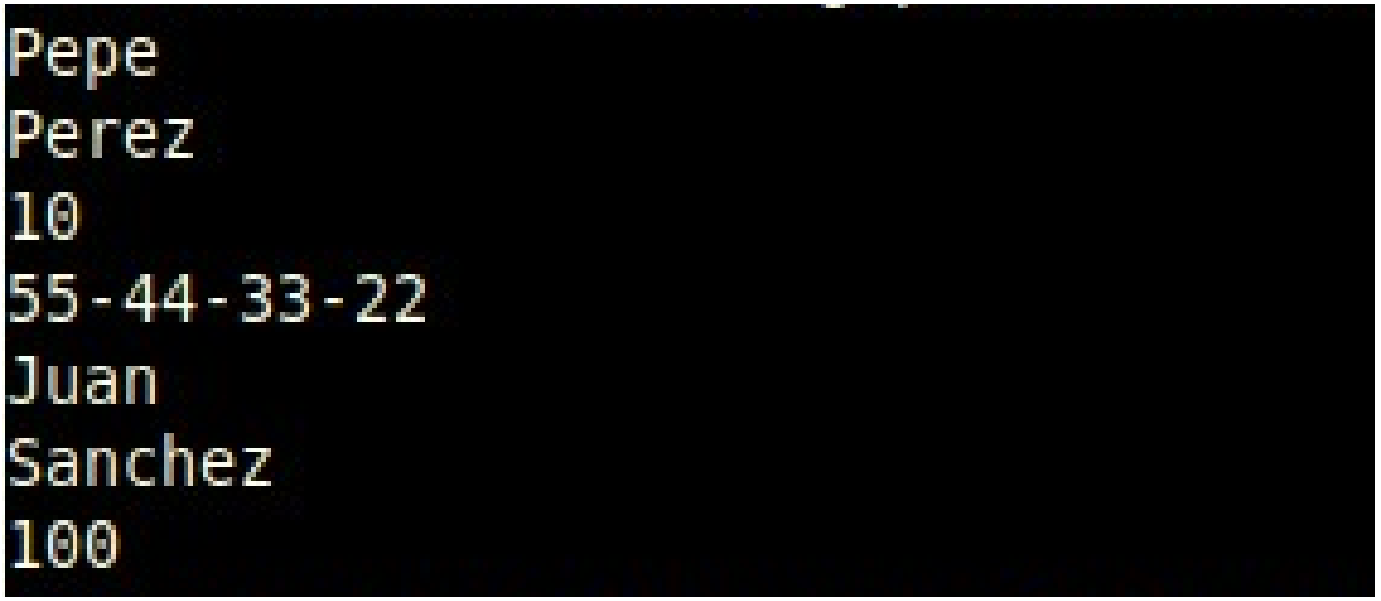
```java
public class User {
  private final String firstName; // required
  private final String lastName; // required
  private final int age; // optional
  private final String phone; // optional
  private final String address; // optional

  private User(Builder builder) {
    this.firstName = builder.firstName;
    this.lastName = builder.lastName;
    this.age = builder.age;
    this.phone = builder.phone;
    this.address = builder.address;
  }

  public String getFirstName() {
    return firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public int getAge() {
    return age;
  }

  public String getPhone() {
    return phone;
  }

  public String getAddress() {
    return address;
  }
```

```java
public abstract class Builder {
    protected String firstName;
    protected String lastName;
    protected int age;
    protected String phone;
    protected String address;

    public abstract User build();

    public abstract Builder age(int age);

    public abstract Builder phone(String phone);

    public abstract Builder address(String address);
}
```

```java
public static class UserBuilder extends Builder{

  public UserBuilder(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  public UserBuilder age(int age) {
    this.age = age;
    return this;
  }

  public UserBuilder phone(String phone) {
    this.phone = phone;
    return this;
  }

  public UserBuilder address(String address) {
    this.address = address;
    return this;
  }

  public User build() {
    return new User(this);
  }

}
```

```
public static class LegalUserBuilder extends Builder{

  public LegalUserBuilder(String firstName, String lastName, int age) {
    this.firstName = firstName;
    this.lastName = lastName;
    if (age < 18) {
      throw new IllegalStateException("No eres mayor de edad");
    }else{
      this.age=age;
    }
  }

  public LegalUserBuilder age(int age) {
    if (age < 18) {
      throw new IllegalStateException("No eres mayor de edad");
    }
    this.age = age;
    return this;
  }
  public LegalUserBuilder phone(String phone) {
    this.phone = phone;
    return this;
  }

  public LegalUserBuilder address(String address) {
    this.address = address;
    return this;
  }

  public User build() {
    return new User(this);
  }

}
}
```

```
public class PruebaUser{
    public static void main(String [] args){
        Builder b = new User.UserBuilder("Pepe","Perez");
        b = b.age(10);
        b = b.phone("55-44-33-22");
        User u = b.build();
        System.out.println(u.getFirstName());
        System.out.println(u.getLastName());
        System.out.println(u.getAge());
        System.out.println(u.getPhone());

        Builder b1 = new User.LegalUserBuilder("Juan","Sanchez",100);
        User u2 = b1.build();
        System.out.println(u2.getFirstName());
        System.out.println(u2.getLastName());
        System.out.println(u2.getAge());
    }
}
```


Pepe
Perez
10
55-44-33-22
Juan
Sanchez
100