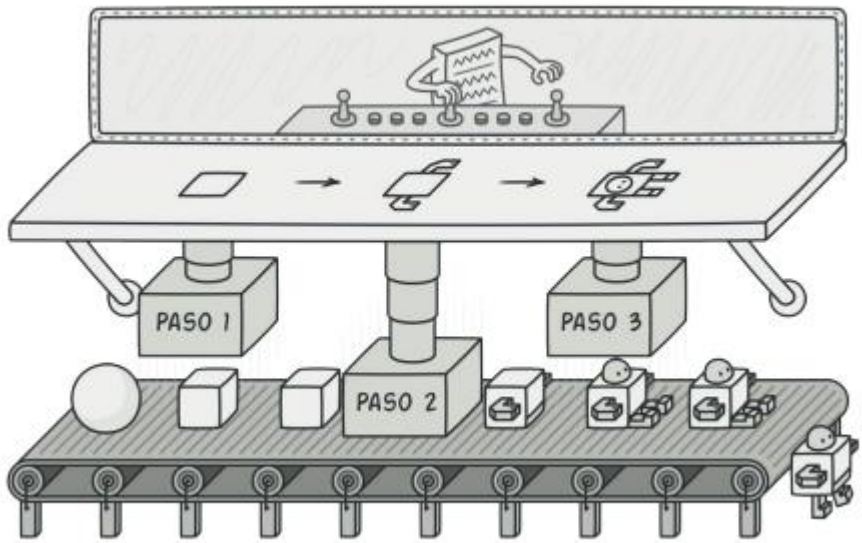
PATRONES DE DISEÑO

1. UN CREACIONAL – BUILDER (CONTRUCTOR)



Se utiliza para construir estructuras complejas con elementos comunes añadiéndole las particularidades.

a. Ejemplo libro: Construcción de autos con sus manuales y 4 componentes o características de producto. (Lenguaje: Java)

builders

builders/Builder.java: Interfaz común del constructor

```
package refactoring_guru.builder.example.builders;

import refactoring_guru.builder.example.cars.CarType;
import refactoring_guru.builder.example.components.Engine;
import refactoring_guru.builder.example.components.GPSNavigator;
import refactoring_guru.builder.example.components.Transmission;
import refactoring_guru.builder.example.components.TripComputer;
```

```java
/**
 * Builder interface defines all possible ways to configure a product.
 */
public interface Builder {
    void setCarType(CarType type);
    void setSeats(int seats);
    void setEngine(Engine engine);
    void setTransmission(Transmission transmission);
    void setTripComputer(TripComputer tripComputer);
    void setGPSNavigator(GPSNavigator gpsNavigator);
}
```

builders/CarBuilder.java: Constructor de auto

```java
package refactoring_guru.builder.example.builders;

import refactoring_guru.builder.example.cars.Car;
import refactoring_guru.builder.example.cars.CarType;
import refactoring_guru.builder.example.components.Engine;
import refactoring_guru.builder.example.components.GPSNavigator;
import refactoring_guru.builder.example.components.Transmission;
import refactoring_guru.builder.example.components.TripComputer;

/**
 * Concrete builders implement steps defined in the common interface.
 */
public class CarBuilder implements Builder {
    private CarType type;
    private int seats;
    private Engine engine;
    private Transmission transmission;
    private TripComputer tripComputer;
    private GPSNavigator gpsNavigator;

    public void setCarType(CarType type) {
        this.type = type;
    }

    @Override
    public void setSeats(int seats) {
        this.seats = seats;
    }

    @Override
    public void setEngine(Engine engine) {
        this.engine = engine;
    }

    @Override
```

```
    public void setTransmission(Transmission transmission) {
        this.transmission = transmission;
    }

    @Override
    public void setTripComputer(TripComputer tripComputer) {
        this.tripComputer = tripComputer;
    }

    @Override
    public void setGPSNavigator(GPSNavigator gpsNavigator) {
        this.gpsNavigator = gpsNavigator;
    }

    public Car getResult() {
        return new Car(type, seats, engine, transmission,
tripComputer, gpsNavigator);
    }
}
```

builders/CarManualBuilder.java: Constructor de manual de auto

```
package refactoring_guru.builder.example.builders;

import refactoring_guru.builder.example.cars.Manual;
import refactoring_guru.builder.example.cars.CarType;
import refactoring_guru.builder.example.components.Engine;
import refactoring_guru.builder.example.components.GPSNavigator;
import refactoring_guru.builder.example.components.Transmission;
import refactoring_guru.builder.example.components.TripComputer;

/**
 * Unlike other creational patterns, Builder can construct unrelated
products,
 * which don't have the common interface.
 *
 * In this case we build a user manual for a car, using the same steps
as we
 * built a car. This allows to produce manuals for specific car
models,
 * configured with different features.
 */
public class CarManualBuilder implements Builder{
    private CarType type;
    private int seats;
    private Engine engine;
    private Transmission transmission;
    private TripComputer tripComputer;
    private GPSNavigator gpsNavigator;
```

```
    @Override
    public void setCarType(CarType type) {
        this.type = type;
    }

    @Override
    public void setSeats(int seats) {
        this.seats = seats;
    }

    @Override
    public void setEngine(Engine engine) {
        this.engine = engine;
    }

    @Override
    public void setTransmission(Transmission transmission) {
        this.transmission = transmission;
    }

    @Override
    public void setTripComputer(TripComputer tripComputer) {
        this.tripComputer = tripComputer;
    }

    @Override
    public void setGPSNavigator(GPSNavigator gpsNavigator) {
        this.gpsNavigator = gpsNavigator;
    }

    public Manual getResult() {
        return new Manual(type, seats, engine, transmission,
tripComputer, gpsNavigator);
    }
}
```

cars

cars/Car.java: Producto auto

```
package refactoring_guru.builder.example.cars;

import refactoring_guru.builder.example.components.Engine;
import refactoring_guru.builder.example.components.GPSNavigator;
import refactoring_guru.builder.example.components.Transmission;
import refactoring_guru.builder.example.components.TripComputer;
```

```java
/**
 * Car is a product class.
 */
public class Car {
    private final CarType carType;
    private final int seats;
    private final Engine engine;
    private final Transmission transmission;
    private final TripComputer tripComputer;
    private final GPSNavigator gpsNavigator;
    private double fuel = 0;

    public Car(CarType carType, int seats, Engine engine, Transmission
transmission,
               TripComputer tripComputer, GPSNavigator gpsNavigator) {
        this.carType = carType;
        this.seats = seats;
        this.engine = engine;
        this.transmission = transmission;
        this.tripComputer = tripComputer;
        if (this.tripComputer != null) {
            this.tripComputer.setCar(this);
        }
        this.gpsNavigator = gpsNavigator;
    }

    public CarType getCarType() {
        return carType;
    }

    public double getFuel() {
        return fuel;
    }

    public void setFuel(double fuel) {
        this.fuel = fuel;
    }

    public int getSeats() {
        return seats;
    }

    public Engine getEngine() {
        return engine;
    }

    public Transmission getTransmission() {
        return transmission;
    }
```

```
    public TripComputer getTripComputer() {
        return tripComputer;
    }

    public GPSNavigator getGpsNavigator() {
        return gpsNavigator;
    }
}
```

cars/Manual.java: Producto manual

```
package refactoring_guru.builder.example.cars;

import refactoring_guru.builder.example.components.Engine;
import refactoring_guru.builder.example.components.GPSNavigator;
import refactoring_guru.builder.example.components.Transmission;
import refactoring_guru.builder.example.components.TripComputer;

/**
 * Car manual is another product. Note that it does not have the same
ancestor
 * as a Car. They are not related.
 */
public class Manual {
    private final CarType carType;
    private final int seats;
    private final Engine engine;
    private final Transmission transmission;
    private final TripComputer tripComputer;
    private final GPSNavigator gpsNavigator;

    public Manual(CarType carType, int seats, Engine engine,
Transmission transmission,
                  TripComputer tripComputer, GPSNavigator
gpsNavigator) {
        this.carType = carType;
        this.seats = seats;
        this.engine = engine;
        this.transmission = transmission;
        this.tripComputer = tripComputer;
        this.gpsNavigator = gpsNavigator;
    }

    public String print() {
        String info = "";
        info += "Type of car: " + carType + "\n";
        info += "Count of seats: " + seats + "\n";
```

```
        info += "Engine: volume - " + engine.getVolume() + "; mileage
- " + engine.getMileage() + "\n";
        info += "Transmission: " + transmission + "\n";
        if (this.tripComputer != null) {
            info += "Trip Computer: Functional" + "\n";
        } else {
            info += "Trip Computer: N/A" + "\n";
        }
        if (this.gpsNavigator != null) {
            info += "GPS Navigator: Functional" + "\n";
        } else {
            info += "GPS Navigator: N/A" + "\n";
        }
        return info;
    }
}
```

cars/CarType.java

```
package refactoring_guru.builder.example.cars;

public enum CarType {
    CITY_CAR, SPORTS_CAR, SUV
}
```

components

components/Engine.java: Característica de producto 1

```
package refactoring_guru.builder.example.components;

/**
 * Just another feature of a car.
 */
public class Engine {
    private final double volume;
    private double mileage;
    private boolean started;

    public Engine(double volume, double mileage) {
        this.volume = volume;
        this.mileage = mileage;
    }

    public void on() {
        started = true;
    }
```

```
    public void off() {
        started = false;
    }

    public boolean isStarted() {
        return started;
    }

    public void go(double mileage) {
        if (started) {
            this.mileage += mileage;
        } else {
            System.err.println("Cannot go(), you must start engine
first!");
        }
    }

    public double getVolume() {
        return volume;
    }

    public double getMileage() {
        return mileage;
    }
}
```

components/GPSNavigator.java: Característica de producto 2

```
package refactoring_guru.builder.example.components;

/**
 * Just another feature of a car.
 */
public class GPSNavigator {
    private String route;

    public GPSNavigator() {
        this.route = "221b, Baker Street, London  to Scotland Yard, 8-
10 Broadway, London";
    }

    public GPSNavigator(String manualRoute) {
        this.route = manualRoute;
    }

    public String getRoute() {
        return route;
    }
```

```
}
```

components/Transmission.java: Característica de producto 3

```
package refactoring_guru.builder.example.components;

/**
 * Just another feature of a car.
 */
public enum Transmission {
    SINGLE_SPEED, MANUAL, AUTOMATIC, SEMI_AUTOMATIC
}
```

components/TripComputer.java: Característica de producto 4

```
package refactoring_guru.builder.example.components;

import refactoring_guru.builder.example.cars.Car;

/**
 * Just another feature of a car.
 */
public class TripComputer {

    private Car car;

    public void setCar(Car car) {
        this.car = car;
    }

    public void showFuelLevel() {
        System.out.println("Fuel level: " + car.getFuel());
    }

    public void showStatus() {
        if (this.car.getEngine().isStarted()) {
            System.out.println("Car is started");
        } else {
            System.out.println("Car isn't started");
        }
    }
}
```

director

director/Director.java: El director controla los constructores

```
package refactoring_guru.builder.example.director;

import refactoring_guru.builder.example.builders.Builder;
import refactoring_guru.builder.example.cars.CarType;
import refactoring_guru.builder.example.components.Engine;
import refactoring_guru.builder.example.components.GPSNavigator;
import refactoring_guru.builder.example.components.Transmission;
import refactoring_guru.builder.example.components.TripComputer;

/**
 * Director defines the order of building steps. It works with a
builder object
 * through common Builder interface. Therefore it may not know what
product is
 * being built.
 */
public class Director {

    public void constructSportsCar(Builder builder) {
        builder.setCarType(CarType.SPORTS_CAR);
        builder.setSeats(2);
        builder.setEngine(new Engine(3.0, 0));
        builder.setTransmission(Transmission.SEMI_AUTOMATIC);
        builder.setTripComputer(new TripComputer());
        builder.setGPSNavigator(new GPSNavigator());
    }

    public void constructCityCar(Builder builder) {
        builder.setCarType(CarType.CITY_CAR);
        builder.setSeats(2);
        builder.setEngine(new Engine(1.2, 0));
        builder.setTransmission(Transmission.AUTOMATIC);
        builder.setTripComputer(new TripComputer());
        builder.setGPSNavigator(new GPSNavigator());
    }

    public void constructSUV(Builder builder) {
        builder.setCarType(CarType.SUV);
        builder.setSeats(4);
        builder.setEngine(new Engine(2.5, 0));
        builder.setTransmission(Transmission.MANUAL);
        builder.setGPSNavigator(new GPSNavigator());
    }
}
```

Demo.java: Código cliente

```
package refactoring_guru.builder.example;
```

```
import refactoring_guru.builder.example.builders.CarBuilder;
import refactoring_guru.builder.example.builders.CarManualBuilder;
import refactoring_guru.builder.example.cars.Car;
import refactoring_guru.builder.example.cars.Manual;
import refactoring_guru.builder.example.director.Director;

/**
 * Demo class. Everything comes together here.
 */
public class Demo {

    public static void main(String[] args) {
        Director director = new Director();

        // Director gets the concrete builder object from the client
        // (application code). That's because application knows better
which
        // builder to use to get a specific product.
        CarBuilder builder = new CarBuilder();
        director.constructSportsCar(builder);

        // The final product is often retrieved from a builder object,
since
        // Director is not aware and not dependent on concrete
builders and
        // products.
        Car car = builder.getResult();
        System.out.println("Car built:\n" + car.getCarType());


        CarManualBuilder manualBuilder = new CarManualBuilder();

        // Director may know several building recipes.
        director.constructSportsCar(manualBuilder);
        Manual carManual = manualBuilder.getResult();
        System.out.println("\nCar manual built:\n" +
carManual.print());
    }

}
```

b. Otro Ejemplo: Datos de cuenta bancaria (lenguaje Java)

//clase normal

package patterns.builder;

```
public class BankAccount {

    private long accountNumber;
    private String owner;
    private BankAccountType type;
    private double balance;
    private double interestRate;

    public BankAccount() {
    }

    public long getAccountNumber() {
        return accountNumber;
    }

    public void setAccountNumber(long accountNumber) {
        this.accountNumber = accountNumber;
    }

    public String getOwner() {
        return owner;
    }

    public void setOwner(String owner) {
        this.owner = owner;
    }

    public BankAccountType getType() {
        return type;
    }

    public void setType(BankAccountType type) {
        this.type = type;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }
```

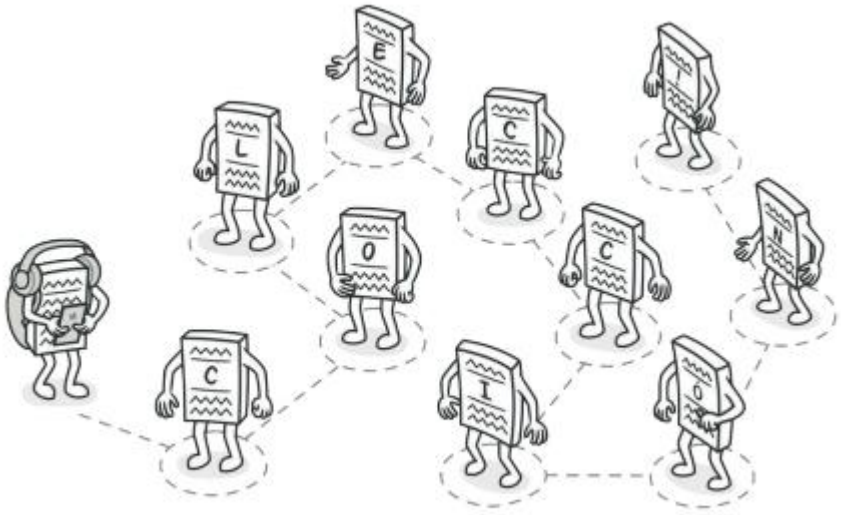
```
        public double getInterestRate() {
            return interestRate;
        }

        public void setInterestRate(double interestRate) {
            this.interestRate = interestRate;
        }
    }

    //Interfaz de Builder  (método Build)

    package patterns.builder;

    public interface IBuilder {
        BankAccount build();
    }
```

```
//implementación de las características (métodos)
package patterns.builder;


public class BankAccountBuilder implements IBuilder {


    private long accountNumber; //This is important, so we will pass it to the constructor.
    private String owner;
    private BankAccountType type;
    private double balance;
    private double interestRate;


    public BankAccountBuilder(long accountNumber) {
        this.accountNumber = accountNumber;
    }


    public BankAccountBuilder withOwner(String owner){
        this.owner = owner;
```

```
        return this;  //By returning the builder each time, we can create a fluent interface.
    }

    public BankAccountBuilder withType(BankAccountType type){
        this.type = type;
        return this;
    }

    public BankAccountBuilder withBalance(double balance){
        this.balance = balance;
        return this;
    }

    public BankAccountBuilder withRate(double interestRate){
        this.interestRate = interestRate;
        return this;
    }

    @Override
    public BankAccount build(){
        BankAccount account = new BankAccount();
        account.setAccountNumber(this.accountNumber);
        account.setOwner(this.owner);
        account.setType(this.type);
        account.setBalance(this.balance);
        account.setInterestRate(this.interestRate);
        return account;
    }
}
```

```
//Creacion del objeto con paso de parámetros y llamado al patron Builder

package patterns.builder;

public class BuilderPatternExample {

    public static void main(String[] args) {

        BankAccountBuilder builder = new BankAccountBuilder(12345l);

        BankAccount bankAccount = builder.withBalance(1000.20)
                .withOwner("Oaken")
                .withRate(10.15)
                .withType(BankAccountType.PLATINUM)
                .build();

        System.out.println(bankAccount);
    }
}

//Datos ejemplo de salida
BankAccount{accountNumber=12345, owner='Oaken', type=PLATINUM, balance=1000.2,
interestRate=10.15}
```

2. UN COMPORTAMENTAL– ITERATOR (ITERADOR)



Permite recorrer elementos sin visualizar su estructura (lista, pilas, árbol, arrays, grafos, cadenas de caracteres, etc.)

a. Ejemplo libro: Conceptual sobre la estructura de este patrón (Lenguaje Python)

main.py: Ejemplo conceptual

```
from __future__ import annotations
from collections.abc import Iterable, Iterator
from typing import Any, List


"""
To create an iterator in Python, there are two abstract classes from the
built-
in `collections` module - Iterable,Iterator. We need to implement the
`__iter__()` method in the iterated object (collection), and the
`__next__ ()`
method in theiterator.
"""


class AlphabeticalOrderIterator(Iterator):
    """
    Concrete Iterators implement various traversal algorithms. These
classes
    store the current traversal position at all times.
    """
```

```
    """
    `_position` attribute stores the current traversal position. An
iterator may
    have a lot of other fields for storing iteration state, especially
when it
    is supposed to work with a particular kind of collection.
    """
    _position: int = None

    """
    This attribute indicates the traversal direction.
    """
    _reverse: bool = False

    def __init__(self, collection: WordsCollection, reverse: bool =
False) -> None:
        self._collection = collection
        self._reverse = reverse
        self._position = -1 if reverse else 0

    def __next__(self):
        """
        The __next__() method must return the next item in the sequence.
On
        reaching the end, and in subsequent calls, it must raise
StopIteration.
        """
        try:
            value = self._collection[self._position]
            self._position += -1 if self._reverse else 1
        except IndexError:
            raise StopIteration()

        return value


class WordsCollection(Iterable):
    """
    Concrete Collections provide one or several methods for retrieving
fresh
    iterator instances, compatible with the collection class.
    """

    def __init__(self, collection: List[Any] = []) -> None:
        self._collection = collection

    def __iter__(self) -> AlphabeticalOrderIterator:
        """
```

```
        The __iter__() method returns the iterator object itself, by
default we
        return the iterator in ascending order.
        """
        return AlphabeticalOrderIterator(self._collection)

    def get_reverse_iterator(self) -> AlphabeticalOrderIterator:
        return AlphabeticalOrderIterator(self._collection, True)

    def add_item(self, item: Any):
        self._collection.append(item)


if __name__ == "__main__":
    # The client code may or may not know about the Concrete Iterator or
    # Collection classes, depending on the level of indirection you want
to keep
    # in your program.
    collection = WordsCollection()
    collection.add_item("First")
    collection.add_item("Second")
    collection.add_item("Third")

    print("Straight traversal:")
    print("\n".join(collection))
    print("")

    print("Reverse traversal:")
    print("\n".join(collection.get_reverse_iterator()), end="")
```

Output.txt: Resultado de la ejecución

```
Straight traversal:
First
Second
Third

Reverse traversal:
Third
Second
First
```

b. **Otro Ejemplo: Iteraccion en perfiles de redes sociales. (Lenguaje: Java) Recorriendo los 'amigos' y 'colegas' de un perfil**

iterators

iterators/ProfileIterator.java: Define la interfaz del perfil

```
package refactoring_guru.iterator.example.iterators;

import refactoring_guru.iterator.example.profile.Profile;

public interface ProfileIterator {
    boolean hasNext();

    Profile getNext();

    void reset();
}
```

iterators/FacebookIterator.java: Implementa la iteración por perfiles de Facebook

```
package refactoring_guru.iterator.example.iterators;

import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.Facebook;

import java.util.ArrayList;
import java.util.List;

public class FacebookIterator implements ProfileIterator {
    private Facebook facebook;
    private String type;
    private String email;
    private int currentPosition = 0;
    private List<String> emails = new ArrayList<>();
    private List<Profile> profiles = new ArrayList<>();

    public FacebookIterator(Facebook facebook, String type, String email)
{
        this.facebook = facebook;
        this.type = type;
        this.email = email;
    }

    private void lazyLoad() {
        if (emails.size() == 0) {
            List<String> profiles =
facebook.requestProfileFriendsFromFacebook(this.email, this.type);
            for (String profile : profiles) {
                this.emails.add(profile);
                this.profiles.add(null);
            }
        }
```

```
    }

    @Override
    public boolean hasNext() {
        lazyLoad();
        return currentPosition < emails.size();
    }

    @Override
    public Profile getNext() {
        if (!hasNext()) {
            return null;
        }

        String friendEmail = emails.get(currentPosition);
        Profile friendProfile = profiles.get(currentPosition);
        if (friendProfile == null) {
            friendProfile =
facebook.requestProfileFromFacebook(friendEmail);
            profiles.set(currentPosition, friendProfile);
        }
        currentPosition++;
        return friendProfile;
    }

    @Override
    public void reset() {
        currentPosition = 0;
    }
}
```

iterators/LinkedInIterator.java: Implementa la iteración por perfiles de LinkedIn

```
package refactoring_guru.iterator.example.iterators;

import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.LinkedIn;

import java.util.ArrayList;
import java.util.List;

public class LinkedInIterator implements ProfileIterator {
    private LinkedIn linkedIn;
    private String type;
    private String email;
    private int currentPosition = 0;
    private List<String> emails = new ArrayList<>();
    private List<Profile> contacts = new ArrayList<>();
```

```
    public LinkedInIterator(LinkedIn linkedIn, String type, String email)
{
        this.linkedIn = linkedIn;
        this.type = type;
        this.email = email;
    }

    private void lazyLoad() {
        if (emails.size() == 0) {
            List<String> profiles =
linkedIn.requestRelatedContactsFromLinkedInAPI(this.email, this.type);
            for (String profile : profiles) {
                this.emails.add(profile);
                this.contacts.add(null);
            }
        }
    }

    @Override
    public boolean hasNext() {
        lazyLoad();
        return currentPosition < emails.size();
    }

    @Override
    public Profile getNext() {
        if (!hasNext()) {
            return null;
        }

        String friendEmail = emails.get(currentPosition);
        Profile friendContact = contacts.get(currentPosition);
        if (friendContact == null) {
            friendContact =
linkedIn.requestContactInfoFromLinkedInAPI(friendEmail);
            contacts.set(currentPosition, friendContact);
        }
        currentPosition++;
        return friendContact;
    }

    @Override
    public void reset() {
        currentPosition = 0;
    }
}
```

social_networks

social_networks/SocialNetwork.java: Define una interfaz común de red social

```
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.ProfileIterator;

public interface SocialNetwork {
    ProfileIterator createFriendsIterator(String profileEmail);

    ProfileIterator createCoworkersIterator(String profileEmail);
}
```

social_networks/Facebook.java: Facebook

```
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.FacebookIterator;
import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;

import java.util.ArrayList;
import java.util.List;

public class Facebook implements SocialNetwork {
    private List<Profile> profiles;

    public Facebook(List<Profile> cache) {
        if (cache != null) {
            this.profiles = cache;
        } else {
            this.profiles = new ArrayList<>();
        }
    }

    public Profile requestProfileFromFacebook(String profileEmail) {
        // Here would be a POST request to one of the Facebook API
endpoints.
        // Instead, we emulates long network connection, which you would
expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("Facebook: Loading profile '" + profileEmail +
"' over the network...");

        // ...and return test data.
        return findProfile(profileEmail);
    }
```

```
    public List<String> requestProfileFriendsFromFacebook(String
profileEmail, String contactType) {
        // Here would be a POST request to one of the Facebook API
endpoints.
        // Instead, we emulates long network connection, which you would
expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("Facebook: Loading '" + contactType + "' list
of '" + profileEmail + "' over the network...");

        // ...and return test data.
        Profile profile = findProfile(profileEmail);
        if (profile != null) {
            return profile.getContacts(contactType);
        }
        return null;
    }

    private Profile findProfile(String profileEmail) {
        for (Profile profile : profiles) {
            if (profile.getEmail().equals(profileEmail)) {
                return profile;
            }
        }
        return null;
    }

    private void simulateNetworkLatency() {
        try {
            Thread.sleep(2500);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public ProfileIterator createFriendsIterator(String profileEmail) {
        return new FacebookIterator(this, "friends", profileEmail);
    }

    @Override
    public ProfileIterator createCoworkersIterator(String profileEmail) {
        return new FacebookIterator(this, "coworkers", profileEmail);
    }

}
```

social_networks/LinkedIn.java: LinkedIn

```java
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.LinkedInIterator;
import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;

import java.util.ArrayList;
import java.util.List;

public class LinkedIn implements SocialNetwork {
    private List<Profile> contacts;

    public LinkedIn(List<Profile> cache) {
        if (cache != null) {
            this.contacts = cache;
        } else {
            this.contacts = new ArrayList<>();
        }
    }

    public Profile requestContactInfoFromLinkedInAPI(String profileEmail)
{
        // Here would be a POST request to one of the LinkedIn API
endpoints.
        // Instead, we emulates long network connection, which you would
expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("LinkedIn: Loading profile '" + profileEmail +
"' over the network...");

        // ...and return test data.
        return findContact(profileEmail);
    }

    public List<String> requestRelatedContactsFromLinkedInAPI(String
profileEmail, String contactType) {
        // Here would be a POST request to one of the LinkedIn API
endpoints.
        // Instead, we emulates long network connection, which you would
expect
        // in the real life.
        simulateNetworkLatency();
        System.out.println("LinkedIn: Loading '" + contactType + "' list
of '" + profileEmail + "' over the network...");

        // ...and return test data.
        Profile profile = findContact(profileEmail);
        if (profile != null) {
```

```
            return profile.getContacts(contactType);
        }
        return null;
    }

    private Profile findContact(String profileEmail) {
        for (Profile profile : contacts) {
            if (profile.getEmail().equals(profileEmail)) {
                return profile;
            }
        }
        return null;
    }

    private void simulateNetworkLatency() {
        try {
            Thread.sleep(2500);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public ProfileIterator createFriendsIterator(String profileEmail) {
        return new LinkedInIterator(this, "friends", profileEmail);
    }

    @Override
    public ProfileIterator createCoworkersIterator(String profileEmail) {
        return new LinkedInIterator(this, "coworkers", profileEmail);
    }
}
```

profile

profile/Profile.java: Perfiles sociales

```
package refactoring_guru.iterator.example.profile;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Profile {
    private String name;
    private String email;
    private Map<String, List<String>> contacts = new HashMap<>();
```

```
    public Profile(String email, String name, String... contacts) {
        this.email = email;
        this.name = name;

        // Parse contact list from a set of "friend:email@gmail.com"
pairs.
        for (String contact : contacts) {
            String[] parts = contact.split(":");
            String contactType = "friend", contactEmail;
            if (parts.length == 1) {
                contactEmail = parts[0];
            }
            else {
                contactType = parts[0];
                contactEmail = parts[1];
            }
            if (!this.contacts.containsKey(contactType)) {
                this.contacts.put(contactType, new ArrayList<>());
            }
            this.contacts.get(contactType).add(contactEmail);
        }
    }

    public String getEmail() {
        return email;
    }

    public String getName() {
        return name;
    }

    public List<String> getContacts(String contactType) {
        if (!this.contacts.containsKey(contactType)) {
            this.contacts.put(contactType, new ArrayList<>());
        }
        return contacts.get(contactType);
    }
}
```

spammer

spammer/SocialSpammer.java: Aplicación de envío de mensajes

```
package refactoring_guru.iterator.example.spammer;

import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;
```

```
import refactoring_guru.iterator.example.social_networks.SocialNetwork;

public class SocialSpammer {
    public SocialNetwork network;
    public ProfileIterator iterator;

    public SocialSpammer(SocialNetwork network) {
        this.network = network;
    }

    public void sendSpamToFriends(String profileEmail, String message) {
        System.out.println("\nIterating over friends...\n");
        iterator = network.createFriendsIterator(profileEmail);
        while (iterator.hasNext()) {
            Profile profile = iterator.getNext();
            sendMessage(profile.getEmail(), message);
        }
    }

    public void sendSpamToCoworkers(String profileEmail, String message)
{
        System.out.println("\nIterating over coworkers...\n");
        iterator = network.createCoworkersIterator(profileEmail);
        while (iterator.hasNext()) {
            Profile profile = iterator.getNext();
            sendMessage(profile.getEmail(), message);
        }
    }

    public void sendMessage(String email, String message) {
        System.out.println("Sent message to: '" + email + "'. Message
body: '" + message + "'");
    }
}
```

Demo.java: Código cliente

```
package refactoring_guru.iterator.example;

import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.Facebook;
import refactoring_guru.iterator.example.social_networks.LinkedIn;
import refactoring_guru.iterator.example.social_networks.SocialNetwork;
import refactoring_guru.iterator.example.spammer.SocialSpammer;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
```

```
/**
 * Demo class. Everything comes together here.
 */
public class Demo {
    public static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        System.out.println("Please specify social network to target spam
tool (default:Facebook):");
        System.out.println("1. Facebook");
        System.out.println("2. LinkedIn");
        String choice = scanner.nextLine();

        SocialNetwork network;
        if (choice.equals("2")) {
            network = new LinkedIn(createTestProfiles());
        }
        else {
            network = new Facebook(createTestProfiles());
        }

        SocialSpammer spammer = new SocialSpammer(network);
        spammer.sendSpamToFriends("anna.smith@bing.com",
                "Hey! This is Anna's friend Josh. Can you do me a favor
and like this post [link]?");
        spammer.sendSpamToCoworkers("anna.smith@bing.com",
                "Hey! This is Anna's boss Jason. Anna told me you would
be interested in [link].");
    }

    public static List<Profile> createTestProfiles() {
        List<Profile> data = new ArrayList<Profile>();
        data.add(new Profile("anna.smith@bing.com", "Anna Smith",
"friends:mad_max@ya.com", "friends:catwoman@yahoo.com",
"coworkers:sam@amazon.com"));
        data.add(new Profile("mad_max@ya.com", "Maximilian",
"friends:anna.smith@bing.com", "coworkers:sam@amazon.com"));
        data.add(new Profile("bill@microsoft.eu", "Billie",
"coworkers:avanger@ukr.net"));
        data.add(new Profile("avanger@ukr.net", "John Day",
"coworkers:bill@microsoft.eu"));
        data.add(new Profile("sam@amazon.com", "Sam Kitting",
"coworkers:anna.smith@bing.com", "coworkers:mad_max@ya.com",
"friends:catwoman@yahoo.com"));
        data.add(new Profile("catwoman@yahoo.com", "Liza",
"friends:anna.smith@bing.com", "friends:sam@amazon.com"));
        return data;
    }
}
```

OutputDemo.txt: Resultado de la ejecución

```
Please specify social network to target spam tool (default:Facebook):
1. Facebook
2. LinkedIn
> 1

Iterating over friends...

Facebook: Loading 'friends' list of 'anna.smith@bing.com' over the
network...
Facebook: Loading profile 'mad_max@ya.com' over the network...
Sent message to: 'mad_max@ya.com'. Message body: 'Hey! This is Anna's
friend Josh. Can you do me a favor and like this post [link]?'
Facebook: Loading profile 'catwoman@yahoo.com' over the network...
Sent message to: 'catwoman@yahoo.com'. Message body: 'Hey! This is Anna's
friend Josh. Can you do me a favor and like this post [link]?'

Iterating over coworkers...

Facebook: Loading 'coworkers' list of 'anna.smith@bing.com' over the
network...
Facebook: Loading profile 'sam@amazon.com' over the network...
Sent message to: 'sam@amazon.com'. Message body: 'Hey! This is Anna's
boss Jason. A
```