Programozási technológiák

Építő tervezési minta:

Miért preferáljuk az építő tervezési mintát?

Túl sok paramétere lehet a konstruktoroknak nehéz követni a setterek hasonlóak a konstruktorokhoz nem a legjobb megoldás.

Mikor mi bővítjük egy vagy több paraméterrel az osztályunkat létre kell hoznunk egy új konstruktort, így rengeteg konstruktorunk lesz, építő tervezési mintával viszont csak egy új metódust adunk hozzá.

Módosíthatatlan tulajdonságot is célszerű tartani mivel lehet párhuzamos futtatás is, a legjobb megoldás ha az objektumaink nem módosulnak miután mi létrehoztuk őket ők az „immutable objects”

A fent felsorolt érvek miatt használtam a megoldásban építő tervezési mintát a felhasználók és beszállítók tárolására.

public class User {  
 private final String name;  
 private final String email;  
 private final String address;  
 private final int age;  
 private final String nameOfMother;  
 private final String mobilePhone;  
  
 public User(Builder builder){  
 this.name = builder.name;  
 this.email = builder.email;  
 this.address = builder.address;  
 this.age = builder.age;  
 this.nameOfMother = builder.nameOfMother;  
 this.mobilePhone = builder.mobilePhone;  
 }  
  
 public static class Builder {  
  
 private final String name;  
 private final String email;  
 private String address;  
 private int age;  
 private String nameOfMother;  
 private String mobilePhone;  
  
 public Builder(String name, String email){  
 this.name = name;  
 this.email = email;  
 }  
  
 public Builder setAddress(String address){  
 this.address = address;  
 return this;  
 }  
  
 public Builder setNameOfMother(String nameOfMother){  
 this.nameOfMother = nameOfMother;  
 return this;  
 }  
  
 public Builder setAge(int age){  
 this.age = age;  
 return this;  
 }  
  
 public Builder setMobilePhone(String mobilePhone){  
 this.mobilePhone = mobilePhone;  
 return this;  
 }  
  
 public User build(){  
 return new User(this);  
 }  
 }

A név és az email cím megadása kötelező a példányosításnál, a további attribútumokat pedig setterekkel fűzhetjük az objektumunkhoz.

pl:

User user = new User.Builder("Teszt Elek", "teszt\_elek@gmail.com").setAge(28).build();

Ha egy új attribútum kerül a rendszerbe, akkor föl kell venni a User osztályba private láthatósággal és final kulcsszóval, ami változónál konstans-t reprezentál.

A User konstruktorába is föl kell venni, ami egy Builder-t vár paraméterül.

A Builder osztályban szintén szerepel minden változó ami a User osztályunkban azzal a különbséggel, hogy csak a kötelezően megadott változók kapnak konstans kulcsszót.

A Builder konstruktorában csak a kötelező elemek szerepelnek jelen esetben a name és az email.

A setterek mindegyike egy Builder-t ad vissza, jelen példában mindig saját magát fogja vissza adni a this kulcsszóval.

pl:

public Builder setNameOfMother(String nameOfMother){  
 this.nameOfMother = nameOfMother;  
 return this;  
}

A build metódus pedig valójában vissza fogja adni az elkészített User objektumunkat

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

A másik tervezési minta amit használtam az a látogató tervezési minta, ami arra szolgál főként, hogy ne keljen módosítani az objektum struktúránkat ha egy új függvényt szeretnénk hozzáadni.

Ami úgy működik, hogy hozzáadunk egy függvényt, ami elfogadja a visitor osztályt.

Minden Terméket külön osztály reprezentál, ami implementálja a „ShoppingItem” interfészt. Az attribútumai „price”, „name” és „subcontractor” ami a beszállítót hivatott tárolni.

A „ShoppingItem”-nek jelenleg egy implementálandó függvénye van, amivel meghívjuk a „ShoppingVartVisitor” Visit függvényét aminek át adjuk paraméterül az aktuális osztályunkat, itt a „Chair” osztályt.

pl

public class Chair implements ShoppingItem {  
  
 private double price;  
 private String name;  
 private User subcontractor;  
  
 public Chair(String name, double price, User subcontractor) {  
 this.name = name;  
 this.price = price;  
 this.subcontractor = subcontractor;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public void setPrice(double price) {  
 this.price = price;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public double accept(ShoppingCartVisitor visitor) {  
 return visitor.visit(this);  
 }  
}

A „ShoppingVartVisitor”

public interface ShoppingCartVisitor {  
 public double visit(Table table);  
 public double visit(Chair chair);  
}

A „ShoppingItem.accept” függvényben ezeket a függvényeket hívjuk meg, amiket pedig a „ShoppingCart” osztály implementál.

public class ShoppingCart implements ShoppingCartVisitor {  
  
 private User user;  
  
 @Override  
 public double visit(Table table) {  
 return table.getPrice();  
 }  
  
 @Override  
 public double visit(Chair chair) {  
 return chair.getPrice();  
 }  
  
 public User getUser() {  
 return user;  
 }  
  
 public void setUser(User user) {  
 this.user = user;  
 }  
}

Visszaadom a termékek árait az implementációkban és tároljuk a felhasználót, akihez tartozik a bevásárló kosár.

A Main fő osztályomban, létrehozok egy vásárlót:

//Vásárló  
User user = new User.Builder("Teszt Elek", "teszt\_elek@gmail.com").setAge(28).build();

majd egy beszállítót

//Beszállító  
User subcontractorIstvan = new User.Builder("Teszt Istvan", "subcontractor\_istvan@subcontractor.hu").setMobilePhone("+362012345678").build();

Termékeket, amiknek megadok nevet, árat és beszállítót.

//Lista a termékekről  
List<ShoppingItem> items = new ArrayList<>();  
items.add(new Table("table1",22, subcontractorIstvan));  
items.add(new Table("table2",3, subcontractorIstvan));  
items.add(new Table("table3",2,subcontractorIstvan));  
items.add(new Chair("chair1",45,subcontractorIstvan));

Létrehozok még egy bevásárló kosarat, aminek paraméterül adom a vásárlót, amit már létrehoztam

ShoppingCartVisitor shoppingCart = new ShoppingCart();  
((ShoppingCart) shoppingCart).setUser(user);

Majd összeadom a termékek árát.

double sum = 0;  
for(ShoppingItem shoppingItem : items)  
 sum = sum + shoppingItem.accept(shoppingCart);

Legvégül kiíratom a Vásárló nevét és a termékek összegét:

System.*out*.println(((ShoppingCart) shoppingCart).getUser().getName() + " " + sum);