# Projektni obrasci

Observer – TemplateMethod - Facade

# Vremenska stanica

- Vremenska stanica prikuplja podatke o temperaturi, vlažnosti i pritisku i te podatke prosleđuje svim zainteresovanim korisnicima
- Korisnici su displeji za prikaz podataka i svaki od njih može da prikazuje podatke kako god želi
- Postoje tri displeja koji prikazuju podatke o vremenu – jedan jednostavan, jedan koji može da vrši statistička izračunavanja i jedan koji može da predviđa vreme

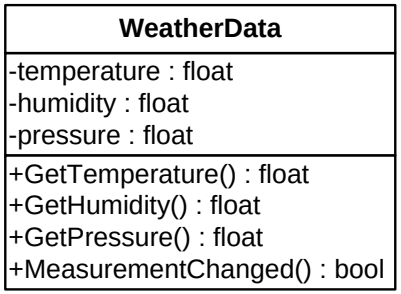

Outdoor Weather
Wind Speed
mph
6.0
Wind Chill
Wind Gust
0
Heat Index
89.8
Dew Point
71.1
Rain Rate
0.00
29.69
064
°F
°C
84.6
RAIN TODAY
Mac
Universal

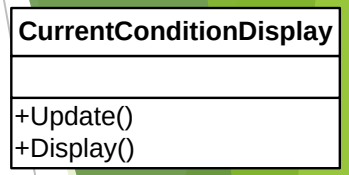

TEMP
OUT
28.0
TEMP
IN
25.0
Oregon

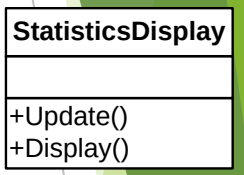

Oregon

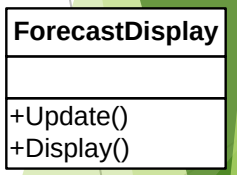
# Loše (najčešće) rešenje

```
public class WeatherData
   {
      CurrentConditionDisplay ccd = new
CurrentConditionDisplay();
      StatisticsDisplay std = new
StatisticsDisplay();
      ForecastDisplay fcd = new
ForecastDisplay();

      public void MeasurementChanged()
      {
         float temp = GetTemperature();
         float press = GetPressure();
         float hum = GetHumidity();

         ccd.Update(temp, hum, press);
         std.Update(temp, hum, press);
         fcd.Update(temp, hum, press);
      }

   }
```



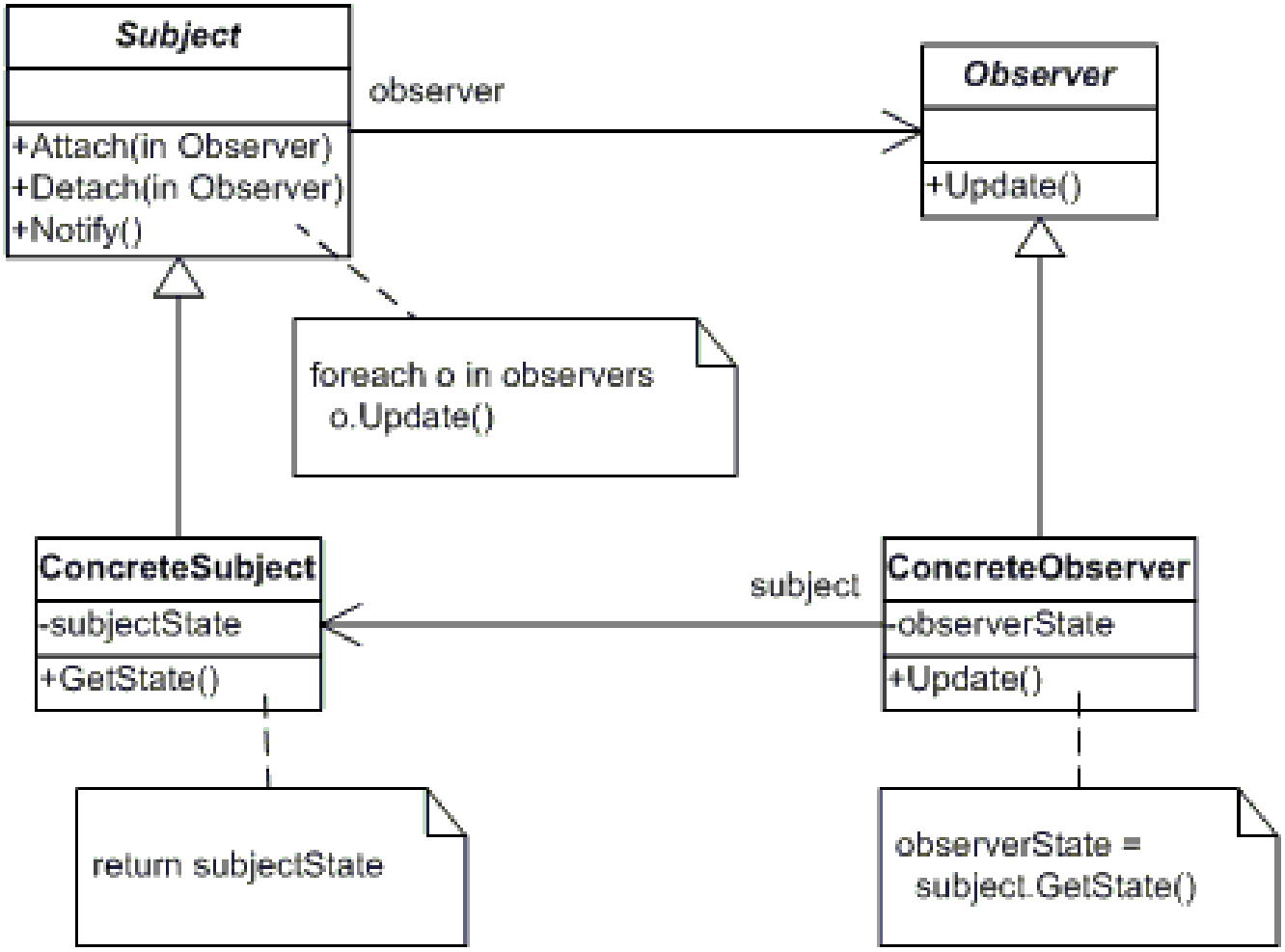



# Definicija

- Definiše zavisnosti tipa jedan-više među različitim objektima i obezbeđuje da se promena stanja u jednom objektu automatski reflektuje u svim zavisnim objektima.
- observer.cs

# UML Dijagram klasa



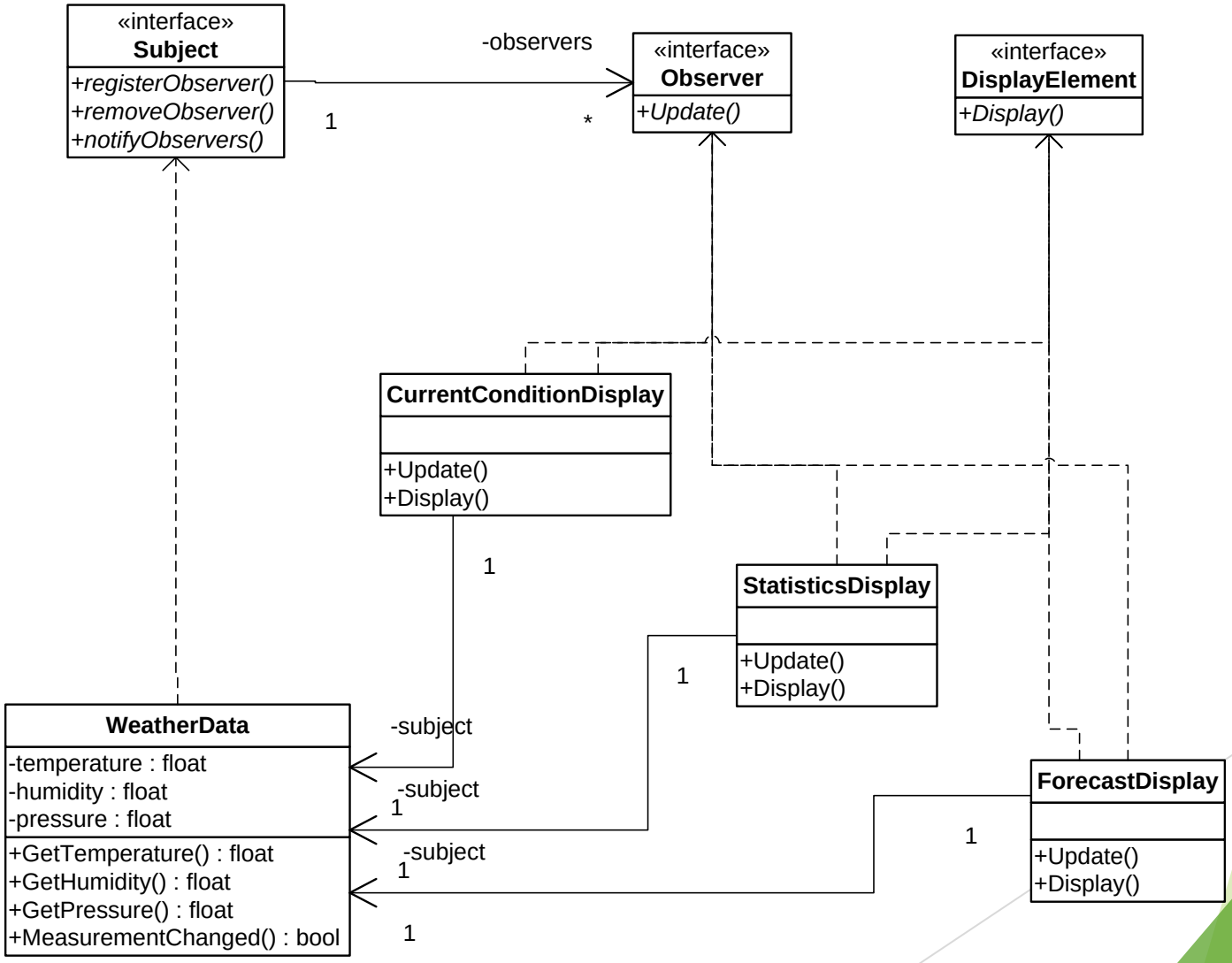
# Elementi

- **Subject (Stock)**
  - čuva referencu prema svom nadzorniku. Jedan objekat može da ima i više Observer objekata.
  - obezbeđuje interfejs za dodavanje i uklanjanje Observer objekata.
- **ConcreteSubject (IBM)**
  - čuva stanje koje će biti od interesa ConcreteObserver objektima
  - šalje notifikaciju svojim nadzornicima kada promeni stanje
- **Observer (IInvestor)**
  - definiše interfejs za ažuriranje objekata nakon što se u objektima klase Subject desi promena
- **ConcreteObserver (Investor)**
  - čuva referencu na ConcreteSubject objekte
  - čuva stanje koje treba da ostane konzistentno sa stanjem roditeljske klase
  - implementira interfejs za ažuriranje objekata koji je definisan u klasi Observer.

# Primer iz stvarnog sveta

- Ovaj primer iz stvarnog sveta prikazuje obrazac Observer u kome se registrovani investitori se obaveštavaju svaki put kada akcije promene svoje vrednosti.
- observerRW.cs

# Primena Observer-a

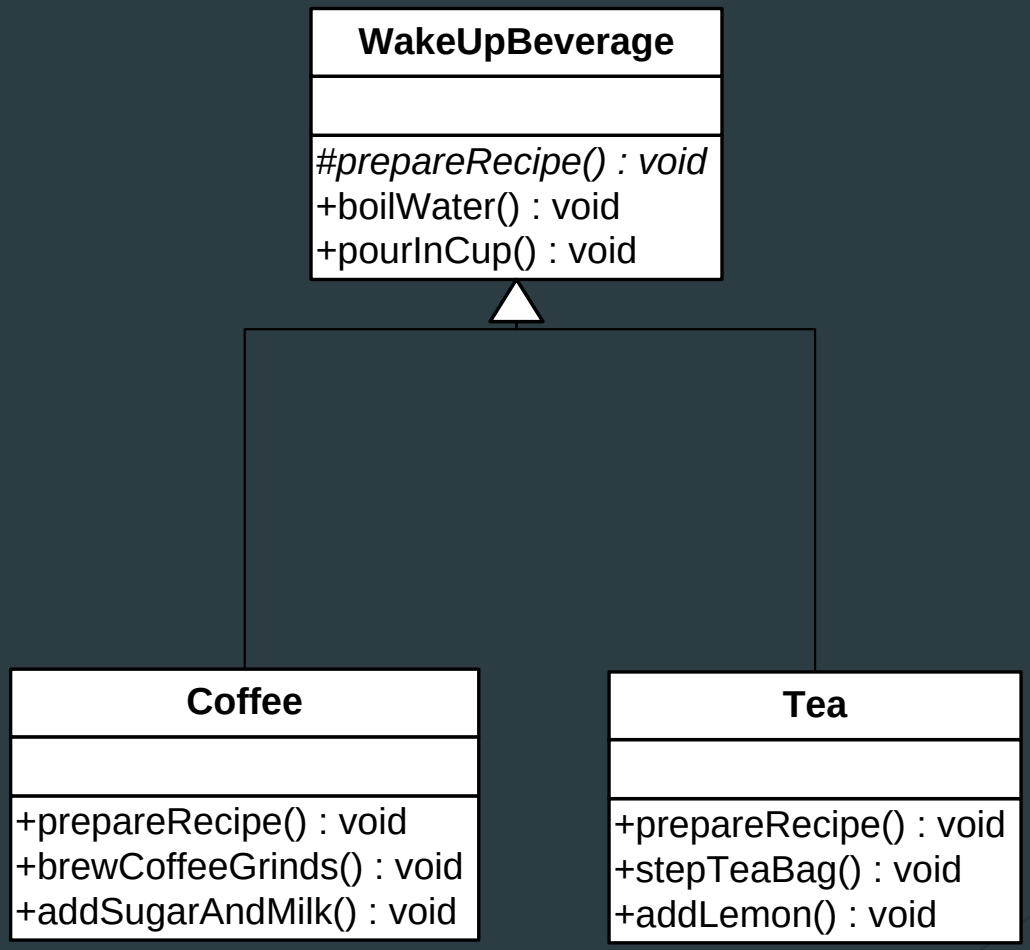


# Kuvanje kafe i čaja – sličnosti, razlike i posledice

```
public class Coffee
   {
      public void PrepareRecipe()
      {
         boilWater();
         brewCoffeeGrinds();
         pourInCup();
         addSugarAndMilk();
      }

      public void addSugarAndMilk()
      {MessageBox.Show("Add sugar & milk");}

      public void pourInCup()
      {MessageBox.Show("Pour coffee in the cup")
}

      public void brewCoffeeGrinds()
      {MessageBox.Show("Brew coffee grinds");}

      public void boilWater()
      {MessageBox.Show("Boil water");       }
   }
```

```
public class Tea
   {
      public void PrepareRecipe()
      {
         boilWater();
         stepTeaBag();
         pourInCup();
         addLemon();
      }

      public void stepTeaBag()
      {MessageBox.Show("AStep tea bag");}

      public void pourInCup()
      {MessageBox.Show("Pour coffee in the cup");}

      public void addLemon()
      {MessageBox.Show("Add lemon");}

      public void boilWater()
      {MessageBox.Show("Boil water");}
   }
```

# Relativno dobro rešenje



```
public abstract class WakeUpBeverage
   {
      public abstract void PrepareRecipe();
      public void boilWater()
      {
         MessageBox.Show("Boil water");
      }
      public void pourInCup()
      {
         MessageBox.Show("Pour coffee in the
cup");
      }
   }
```

```
public class Coffee: WakeUpBeverage
   {
      public override void PrepareRecipe()
      {
         boilWater();
         brewCoffeeGrinds();
         pourInCup();
         addSugarAndMilk();
      }

      public void addSugarAndMilk()
      { MessageBox.Show("Add sugar & milk");}

      public void brewCoffeeGrinds()
      {MessageBox.Show("Brew coffee grinds");}
   }
```

```
public class Tea : WakeUpBeverage
   {
      public override void PrepareRecipe()
      {
         boilWater();
         stepTeaBag();
         pourInCup();
         addLemon();
      }

      public void stepTeaBag()
      {MessageBox.Show("AStep tea bag");}

      public void addLemon()
      { MessageBox.Show("Add lemon");}
   }
```

# Primenjen Template Method

```
public abstract class WakeUpBeverage    {
        public void PrepareRecipe()       {
            boilWater();
            brew();
            pourInCup();
            addCondiments();
        }

        public abstract void addCondiments();
        public abstract void brew();
        public void boilWater()       {
            MessageBox.Show("Boil water");       }
        public void pourInCup()       {
            MessageBox.Show("Pour coffee in thecup");
        }
    }
```

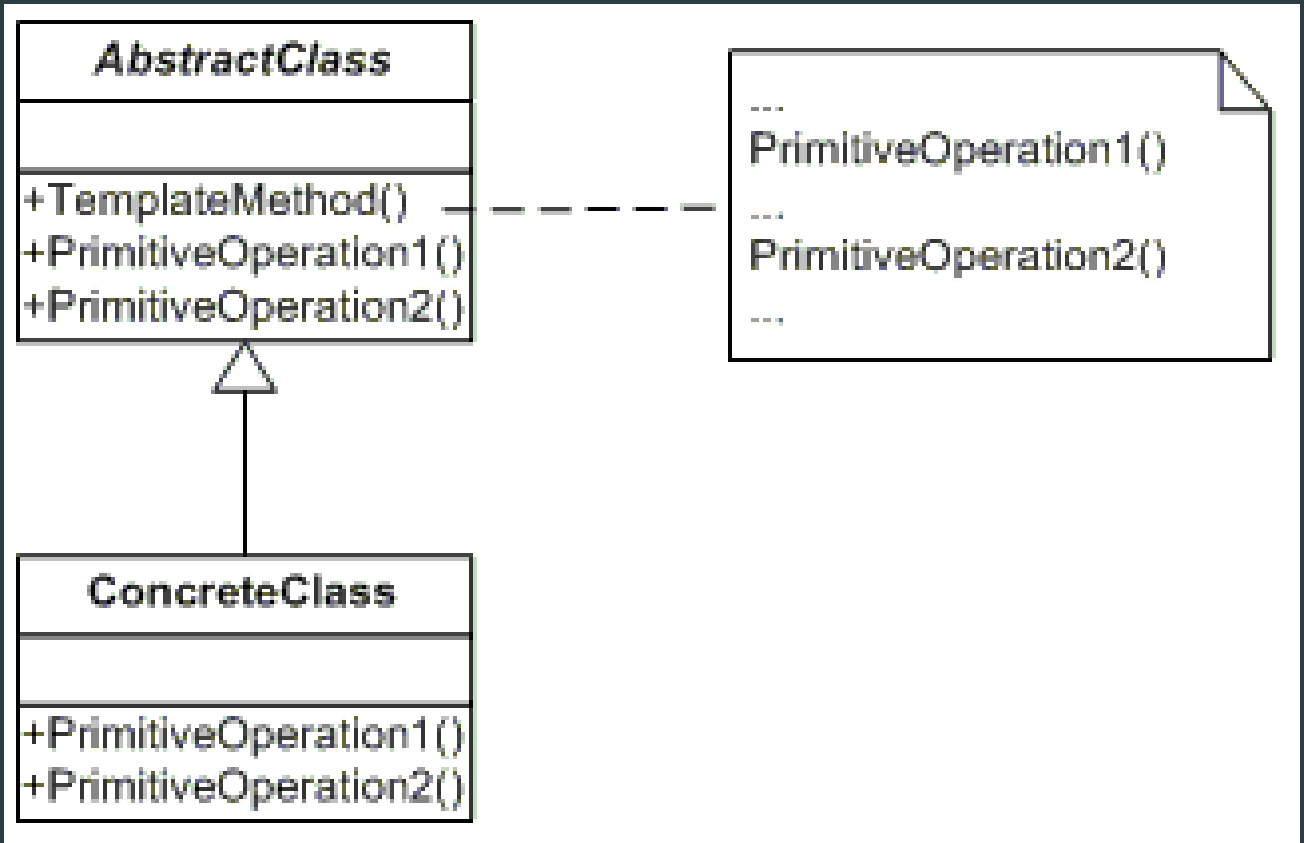
```
public class Coffee: WakeUpBeverage    {
        public void addSugarAndMilk()       {
            MessageBox.Show("Add sugar & milk");
        }
        public void brewCoffeeGrinds()       {
MessageBox.Show("Brew coffee grinds");
        }
        public override void addCondiments()       {
            this.addSugarAndMilk();
        }
        public override void brew()       {
            this.addCondiments();
        }
    }
```

```
public class Tea : WakeUpBeverage    {
        public void stepTeaBag()       {
            MessageBox.Show("AStep tea bag");
        }
        public void addLemon()       {
            MessageBox.Show("Add lemon");
        }
        public override void addCondiments()       {
            this.stepTeaBag();
        }
        public override void brew()       {
            this.addLemon();
        }
    }
```

# Template Method pattern - definicija

- Definiše osnovu algoritma u nekoj operaciji tako što pojedine korake razdvaja u podklase. Ovaj obrazac dozvoljava podklasama da predefinišu određene korake u algoritmu i to bez menjanja strukture algoritma
- templateMethod.cs

# UML Dijagram klasa



# Elementi

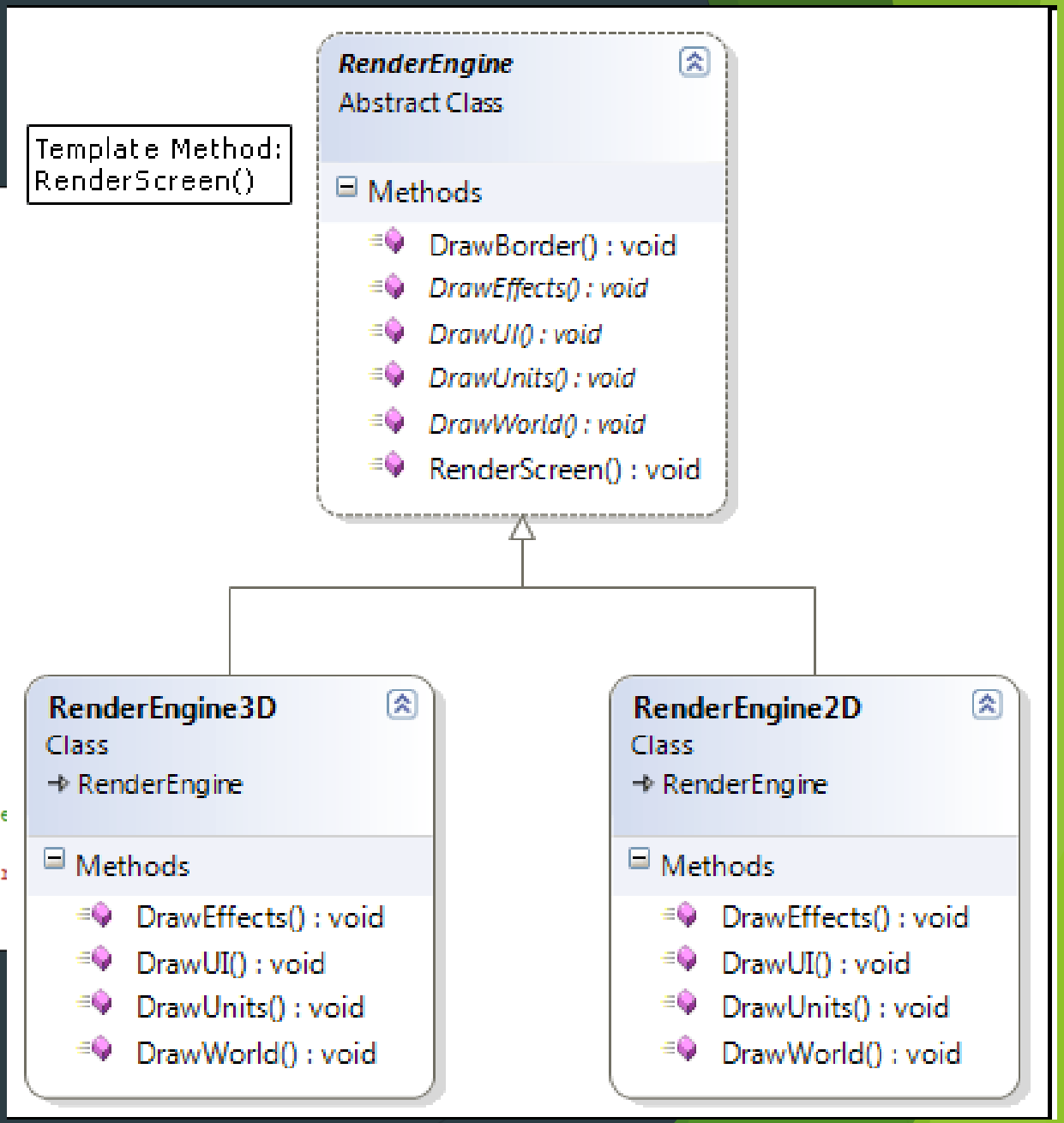
- **AbstractClass (DataObject)**
  - definiše apstraktne primitivne operacije koje konkretne podklase koriste za implementaciju algoritama
  - implementira šablonski metod definišući skelet algoritma. Šablonski metod će pozivati kako primitivne operacije tako i operacije definisane u apstraktnim klasama drugih familija objekata.
- **ConcreteClass (CustomerDataObject)**
  - implementira primitivne operacije i vodi računa o specifičnim koracima u algoritmu koji unose podklase.

# Primer iz stvarnog sveta

- Ovaj primer iz stvarnog sveta pokazuje kako šablonski metod Run() obezbeđuje sekvencu poziva različitih metoda. Implementacije ovih metoda su u klasi CustomerDataObject. Ova podklasa implementira metode Connect, Select, Process i Disconnect.
- templateMethodRW.cs

```
public abstract class RenderEngine
{
    // Don't have 'final' keyword,
    // no such thing in C#
    public void RenderScreen()
    {
        // Rendering algorithm.
        DrawWorld();
        DrawUnits();
        DrawUI();
        DrawEffects();
        DrawBorder();
    }

    public abstract void DrawWorld();
    public abstract void DrawUnits();
    public abstract void DrawUI();
    public abstract void DrawEffects();

    public void DrawBorder()
    {
        // Draw a border around the scre
        // The same for every renderer
        Console.WriteLine("Drawing a bo
    }
}
```

Template Method:
RenderScreen()

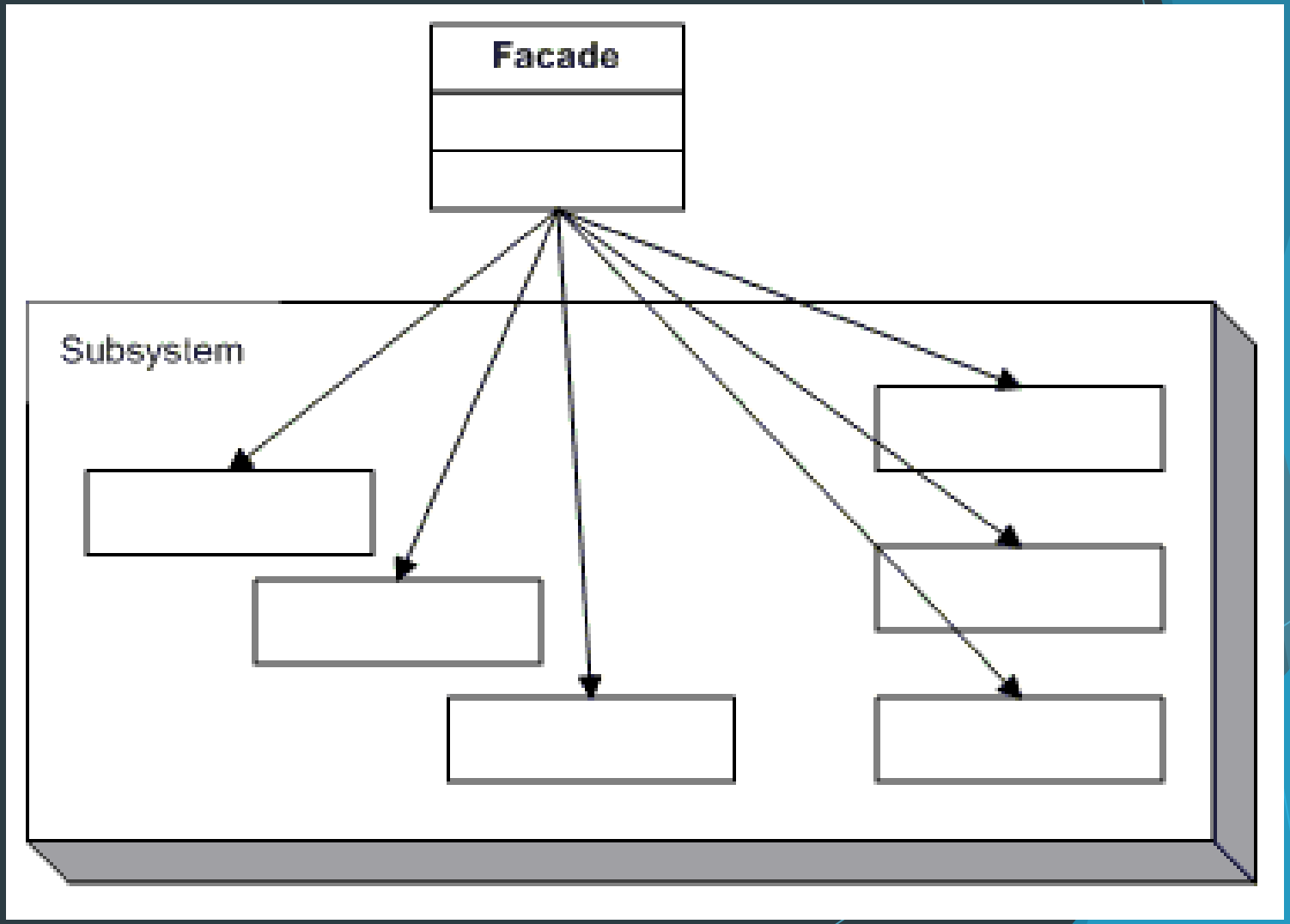


# Façade

# Definicija

- Obezbeđuje jedinstveni interfejs ka celom podsistemu. Obrazac Fasada definiše interfejs na višem nivou koji olakšava upotrebu podsistema.
- facade.cs

# UML Dijagram klasa

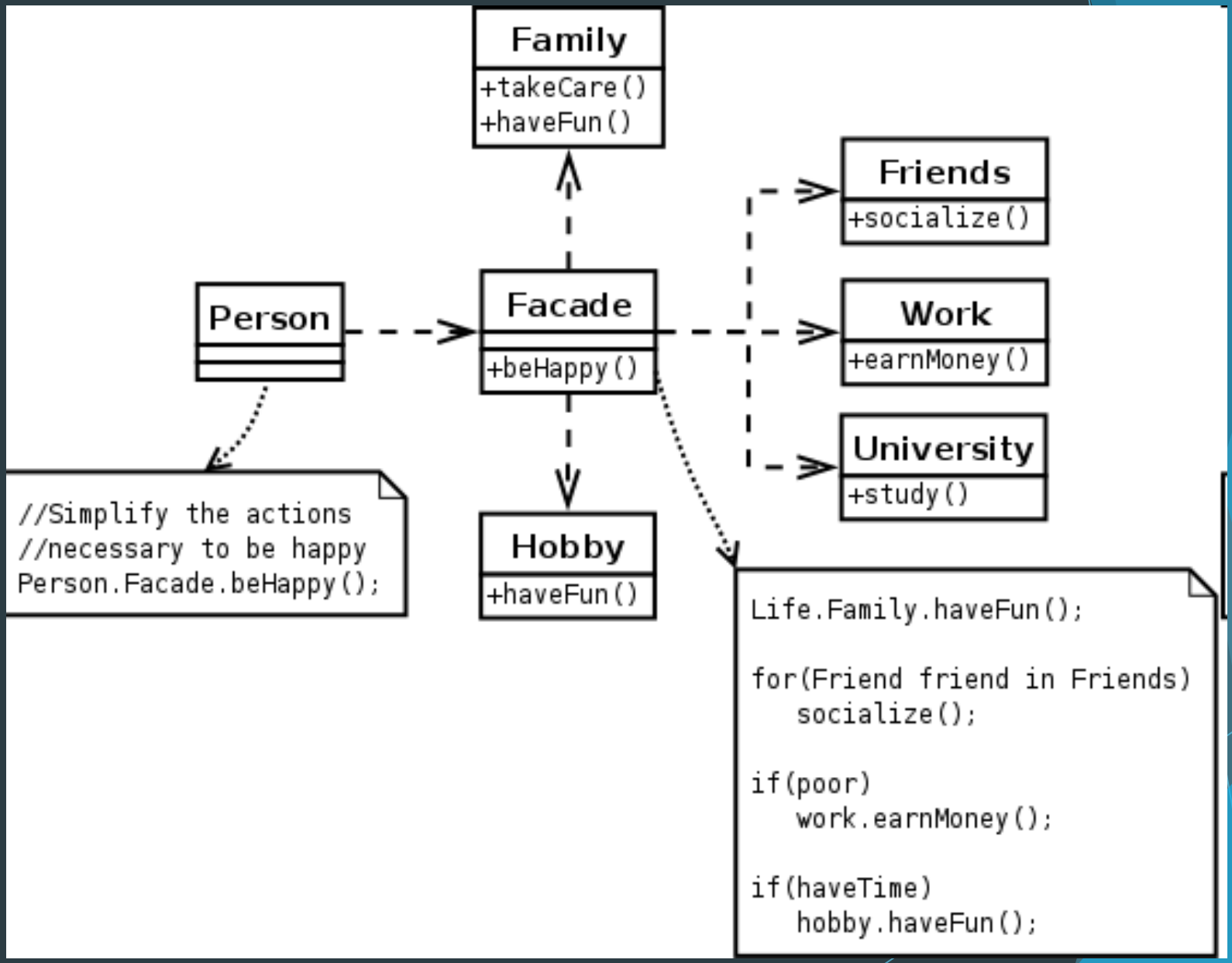


# Elementi

- **Facade (MortgageApplication)**
  - zna koje klase podsistema treba da prihvate spoljni zahtev
  - delegira klijentski zahtev odgovarajućem objektu iz podsistema
- **Subsystem classes (Bank, Credit, Loan)**
  - implementiraju funkcionalnost podsistema
  - obrađuju zahteve koji im dolaze od objekta klase Facade
  - nemaju pojma o fasadi i ne čuvaju nikakvu referencu na nju

# Primer iz stvarnog sveta

- Ovaj primer pokazuje upotrebu obrasca Facade kao objekta klase MortgageApplication (mortgage znači hipoteka) koji obezbeđuje uprošćeni interfejs ka skupu klasa koje imaju ulogu da odrede kreditnu sposobnost klijenta.
- facadeRW.cs


Family
+takeCare()
+haveFun()
Friends
+socialize()
Person
Facade
+beHappy()
Work
+earnMoney()
University
+study()
Hobby
+haveFun()
//Simplify the actions
//necessary to be happy
Person.Facade.beHappy();
Life.Family.haveFun();
for(Friend friend in Friends)
socialize();
if(poor)
work.earnMoney();
if(haveTime)
hobby.haveFun();