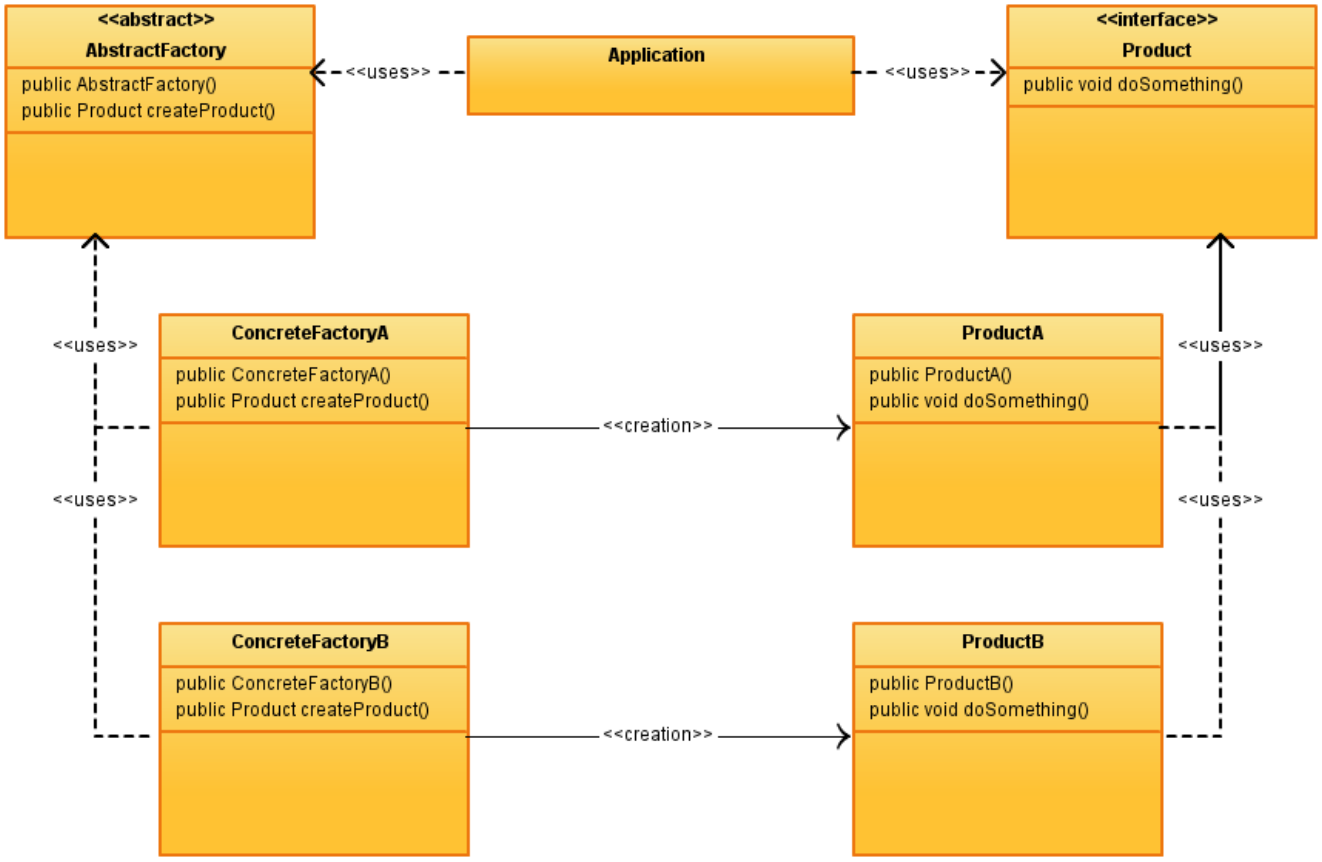
Abstract Factory

Az **Abstract Factory** minta akkor lehet hasznos számunkra, ha egy bizonyos feladatra több implementációt is tudunk használni. Gondoljunk például a Java nyelvtl kissé messze es sörökre, hiszen a sörök alapvet kezelése esetén vélheten azonos mozdulattal kell éljünk, holott a különböz sörök más-más gyárból érkeznek. Az **Abstract Factory** használatával a sörivók ell elrejtjük azt, hogy melyik konkrét sört használja a feladat végrehajtására, egyszeren csak a feladat megoldására kell koncentrálni.

A probléma

Programjaink tervezése során sok esetben futunk bele abba a problémába, hogy a környezet által meghatározott feltételek szerint kell egy-egy feladatot megoldani. Az **Abstract Factory** minta használata feltételezi, hogy a kérdéses feladat végrehajtása eltt el tudjuk dönteni, hogy a környezet milyen konkrét implementációt igényel, hiszen a kliens feladata a megfelel implementáció felhasználása. Ez a gyakorlatban azt jelenti, hogy ismernünk kell a sör nevét, amelybl aztán kérünk egy példányt.

A megoldás



Az objektum orientált programozási módszertan önmagában lehetvé teszi, hogy a kliens program jelents átírása nélkül képesek legyünk egy használt implementációt egy másik implementációval kiváltani:

Java forrás

```
Sör sör = new SoproniÁszokSör();
System.out.println(sör.getNév());
// ...
sör.nyit();
// ...
System.out.println(sör.getNév());
```

A konkrét sört pillanatok alatt kicserélhetjük egy másik fajta sörre:

Java forrás

```
Sör sör = new PécsiSzalonSör();
System.out.println(sör.getNév());
// ...
sör.nyit();
// ...
System.out.println(sör.getNév());
```

A fenti program helyes mködéséhez az kell, hogy mindegyik *konkrét* sör implementálja a *Sör* alapvet tulajdonságait és metódusait, hiszen a program további részein már nem kell tudnunk, hogy éppen soproni vagy pécsi sörgyár termékét szopogatjuk. Ezzel a *Sör* osztállyal az **Abstract Factory** egyik felét meg is oldottuk, hiszen a *Sör* célszeren absztrakt osztály vagy egy interfész, amelynek biztos van egy *nyit* és egy *getNév* metódusa, s ezt implementálja a példában említett két konkrét sör:

Sör.java

```
public interface Sör
{
  public String getNév();
  public void nyit();
  // ...
}
```

Az **Abstract Factory** második fele feltételezi, hogy minden gyár többféle sört képes gyártani, azonban ezek tipizálhatók. Tegyük fel, hogy mindegyik sörgyár képes Pilzeni (világos), Bak (barna) és Búzasört gyártani. Az építkezést az alapoknál kell kezdeni, ezért le kell fektetnünk egy *teljesen átlagos sörgyár* alapjait:

SörGyár.java

```
public abstract class SörGyár
{
  public abstract Sör gyártPilzenit();
  public abstract Sör gyártBakot();
  public abstract Sör gyártBúzasört();
  public abstract Sör gyártPrémiumot();
}
```

Mint látjuk, a *SörGyár* létrehozná a terméket, ha nem lenne absztakt mind a négy *gyárt* metódusa, így ebben az esetben csak a leszármazott képes sört gyártani, hozzuk létre a Pécsi Sörgyárat:

PécsiSörGyár.java

```
public class PécsiSörGyár extends SörGyár
{
  public Sör gyártPilzenit()
  {
    return new PécsiSzalonSör();
  }

  public Sör gyártBakot()
  {
    return new PécsiSzalonBarnaSör();
  }

  public Sör gyártBúzasört()
  {
    return new PécsiSzalonBúzaSör();
  }

  public Sör gyártPrémiumot()
  {
    return new GoldFasslSör();
  }
}
```

Majd a Soproni Sörgyárat (amely egy ideje a Heineken Hungária névre hallgat):

SoproniSörGyár.java

```
public class SoproniSörGyár extends SörGyár
{
  public Sör gyártPilzenit()
  {
    return new SoproniÁszokSör();
  }

  public Sör gyártBakot()
  {
    return new SoproniFeketeDémonSör();
  }

  public Sör gyártBúzasört()
  {
    return new EdelweissSör();
  }

  public Sör gyártPrémiumot()
  {
    return new HeinekenSör();
  }
}
```

Észre kell vennünk, hogy például a leszármazott *PécsiSörGyár* osztály négy *gyárt* metódusa alapján ad vissza konkrét *Sör* osztályokat. Az **Abstract Factory** minta használata során létre kell hoznunk a gyárat, majd a gyár legyártja nekünk a kért terméket:

Java forrás

```
SörGyár sörGyár = new PécsiSörGyár();
Sör sör = sörGyár.gyártBúzasört();
System.out.println(sör.getNév());
// ...
sör.nyit();
// ...
System.out.println(sör.getNév());
```

Mint láthatjuk, a programunk már nem tudja, hogy a *PécsiSzalonBúzaSör* osztály egy példányát használja, a kliens programban nincs is hivatkozás erre az osztályra, a gyár ugyanis egyszeren egy *Sör* interfészt ad vissza, ahogy ezt a sörgyárak teszik, a benne lév referencia azonban már a *PécsiSzalonBúzaSör* példányra mutat.

Elnyök és hátrányok

Az **Abstract Factory** minta használata akkor hasznos, ha a probléma megoldásához fel tudunk használni egy olyan absztrakt osztályt, amely az egyes implementációk se tud lenni, és a jövben nem várható az egyes implementációk közötti különbség megjelenése, vagyis nem jelennek meg olyan sörgyárak, amelyek egyedi sörtípust is gyártanak. Ez esetben ugyanis az **összes** sörgyárnak fel kell vennie ezt a sörtípust is a gyártott sörök közé, hiszen a kliens nem találkozik közvetlenül az adott sörtípussal, csak az sör interfészével, illetve az általánosított sörgyárral, csak azon keresztül tudná az újabb típusú sört legyártatni.

Igen nagy elnye ennek a mintának, hogy a kliens átírása nélkül tudunk konkrét sörtípust váltani, a feladat csak annyi, hogy a kliensnek látnia kell az újabb típusú sör osztályát, de ez nem fordítás idej probléma. A sörgyár programozójának teljesen szabad keze van a konkrét Sör-implementációk cserélgetésében, amíg a Sör interfész minden lényeges tulajdonságát implementálja.

(forrás: wikipedia)