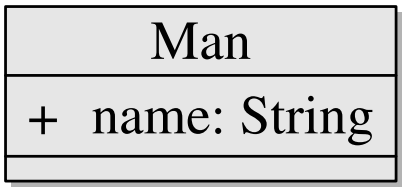
Clase Abstracte si Interfete

Alexandru Olteanu

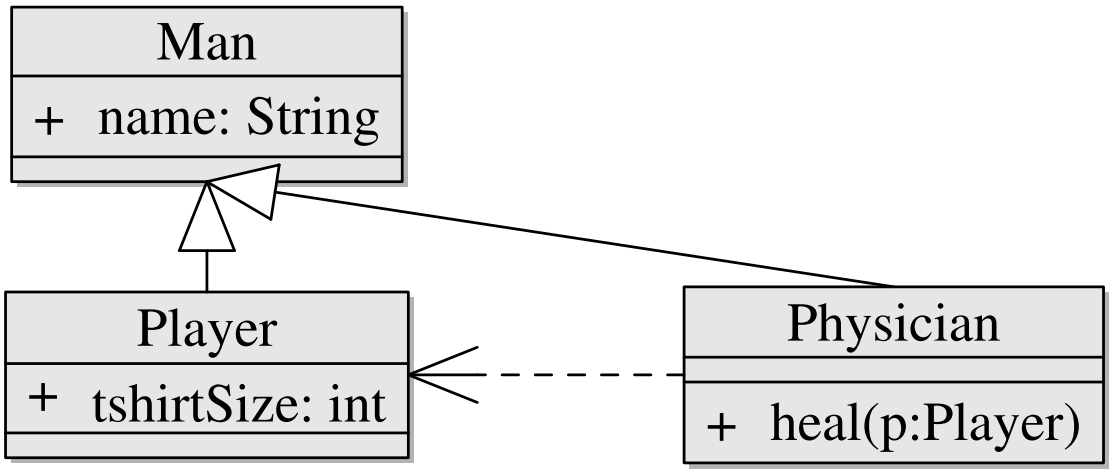
Universitatea Politehnica Bucuresti
Facultatea de Automatică si Calculatoare, Departamentul Calculatoare
alexandru.olteanu@upb.ro

OOP, 2020

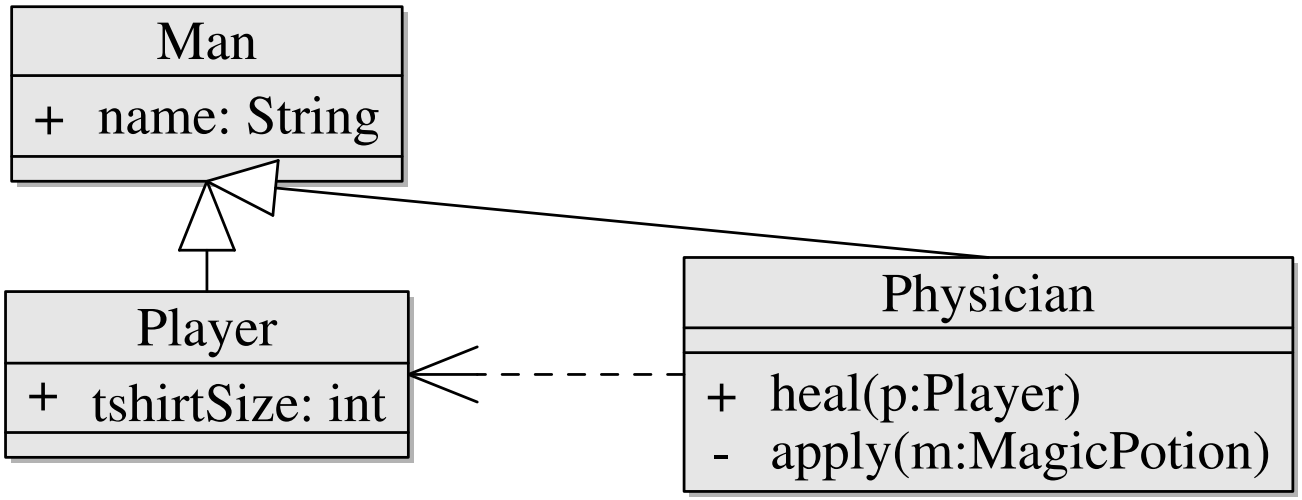
Curs 2: ce contine o clasa



Curs 3: relatia dintre clase

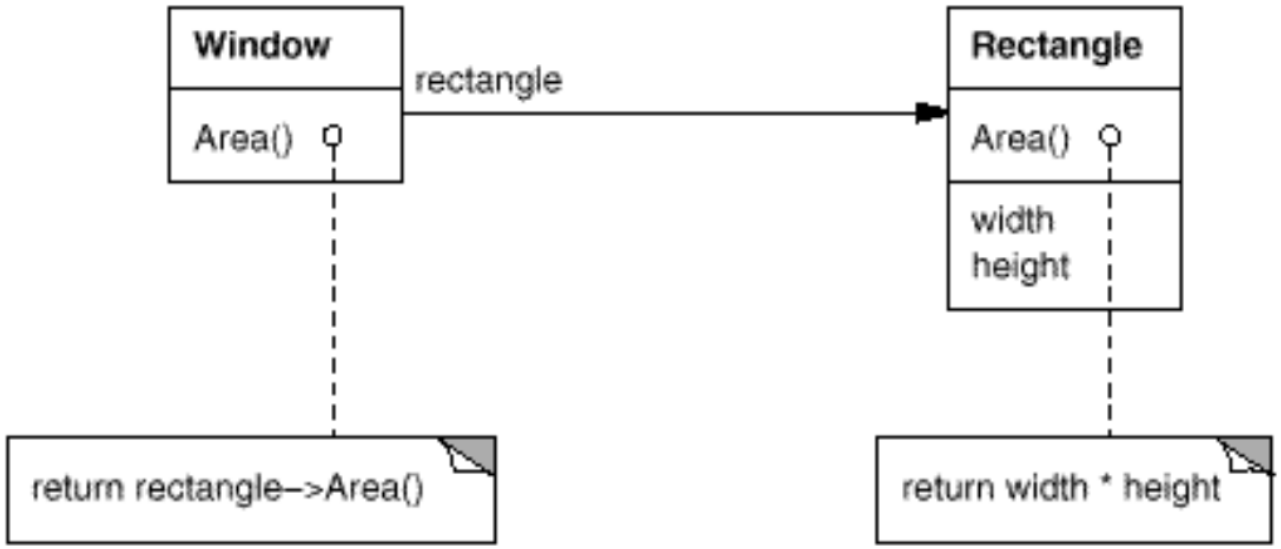


Curs 4: incapsulare, static, final



Delegation example

Pentru a putea schimba implementarea la run-time, avem nevoie de interfete



Delegation, Introduction, "Design Patterns", E. Gamma, R. Helm, R. Johnson, J. Vlissides (aka Gang of Four)

Interfete

Exemplu de interfata cu utilizatorul

Ce este o interfata?

Ganditi-va la o aplicatie care monitorizeaza activitatea angajatilor unei firme

```
public interface Payee {
  int computeMoney(Month paymentMonth);
}
public class Engineer implements Payee {
  private Contract contract;
  public int computeMoney(Month paymentMonth) {
    if (daysWorked + paidLeave == paymentMonth.
        getWorkingDays())
      return contract.getSalary();
  }
}
public class SalesRep implements Payee {
  private Contract contract;
  private int newCustomers;
  public int computeMoney(Month paymentMonth) {
    return contract.getSalary() + newCustomers * contract.
        getNewCustomerBonus();
  }
```

Ce este o interfata?

Ganditi-va la o aplicatie care monitorizeaza activitatea angajatilor unei firme
Din punct de vedere payroll, aplicatia va functiona luna de luna, indeferent cati si ce fel de angajati are firma.

```
public class FinancialManager {
  private BankAccount ba;
  private List<Payee> payroll;
  public executePayroll(Month paymentMonth) {
    for (Payee payee : payroll) {
      ba.pay(payee.computeMoney(paymentMonth));
    }
  }
}
```

Ce este o interfata?

Definition

Interfata este un tip de date care reprezinta, in forma sa de baza, o grupare de semnaturi de metode (fara implementari) unite de un scop comun.

Interfata este un contract care defineste interactiunea a doua componente software. Se foloseste la:

- schimbarea dinamica a componentelor

Ce este o interfata?

Ganditi-va ca vi se cere extinderea aplicatiei pentru un spital

```
public interface Payee {
  int computeMoney(Month paymentMonth);
}
public class Nurse implements Payee {
  private Contract contract;
  public int computeMoney(Month paymentMonth) {
    if (daysWorked + paidLeave == paymentMonth.
        getWorkingDays())
      return contract.getSalary();
  }
}
public class Doctor implements Payee {
  private Contract contract;
  public int computeMoney(Month paymentMonth) {
    return contract.getSalary() + guardShifts * contract.
        getGuardShiftBonus();
  }
}
```

Ce este o interfata?

Ganditi-va ca vi se cere extinderea aplicatiei pentru un spital
Din punct de vedere payroll, aplicatia ramane neschimbata

```
public class FinancialManager {
  private BankAccount ba;
  private List<Payee> payroll;
  public executePayroll(Month paymentMonth) {
    for (Payee payee : payroll) {
      ba.pay(payee.computeMoney(paymentMonth));
    }
  }
}
```

Ce este o interfata?

Definition

Interfata este un tip de date care reprezinta, in forma sa de baza, o grupare de semnaturi de metode (fara implementari) unite de un scop comun.

Interfata este un contract care defineste interactiunea a doua componente software. Se foloseste la:

- schimbarea dinamica a componentelor
- decuplarea si reutilizarea componentelor

Ce poate contine o interfata?

- semnaturi de metode

```
public interface Payee {
   int calcMoney(Month paymentMonth);
}
```

Toate sunt public in mod automat (nu se trec specificatori de acces)

Ce poate contine o interfata?

- semnaturi de metode
- metode statice (inclusiv implementari)
- constante (static final)

```
public interface Payee {
   static final String PAYER = "SuperStar Basket Club";
   static final Bank PAYING_BANK = "Rich Swiss Bank"
   static String printPayerData() {
     System.out.println(PAYING_BANK+" on the behalf of "+
         PAYER);
   }
}
```

Toate sunt public in mod automat (nu se trec specificatori de acces)

Ce poate contine o interfata?

- semnaturi de metode
- metode statice (inclusiv implementari)
- constante (static final)
- metode default (inclusiv implementari) din Java 8, implicatii asupra mostenirii multiple

```
public interface Payee {
  default Contract signContract() {
    return new Contract("Payer Inc.");
  }
}
```

Toate sunt public in mod automat (nu se trec specificatori de acces)

Interfete celebre: List

```
List<CEO> topCEOs = new ArrayList<CEO>();
topCEOs.add(new CEO("Steve Jobs"));
topCEOs.add(new CEO("Bill Gates"));
topCEOs.add(new CEO("Jeff Bezos"));
topCEOs.add(new CEO("Elon Musk"));
```

Interfete celebre: Comparable

```
public class CEO implements Comparable {
    public int netWorth;
    public int compareTo(CEO otherCEO) {
        return netWorth - otherCEO.netWorth();
    }
}
. . .
Collections.sort(topCEOs);
```

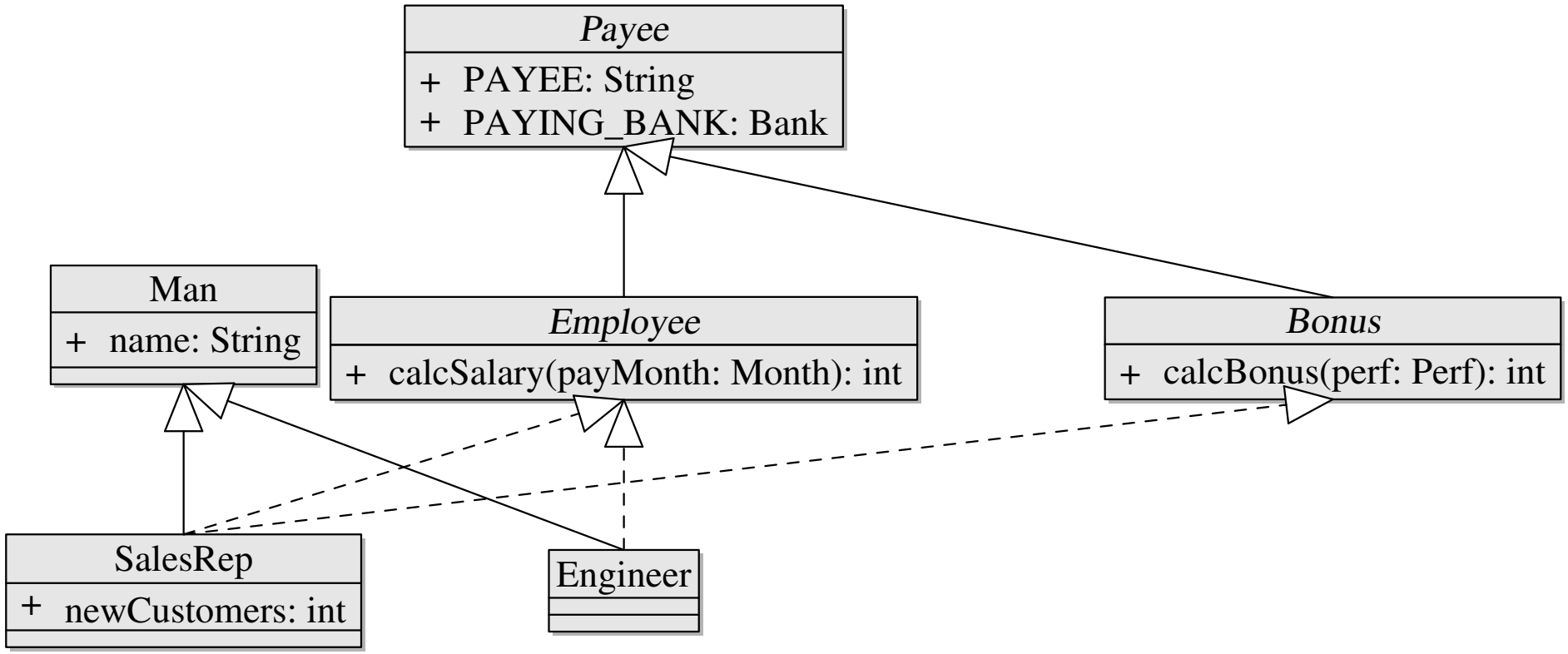
Evolving interfaces

```
public interface Payee {
  static final String PAYER = "SuperStar Basket Club";
  static final Bank PAYING_BANK = "Rich Swiss Bank"
  static String printPayerData() {
    System.out.println(PAYING_BANK+" on the behalf of "+
        PAYER);
  }
  default Contract signContract() {
    return new Contract(PAYER);
  }
}

public interface Employee extends Payee {
  int computeSalary(Month paymentMonth);
}

public interface Bonus extends Payee {
  int computeBonus(Performance performance);
}
```

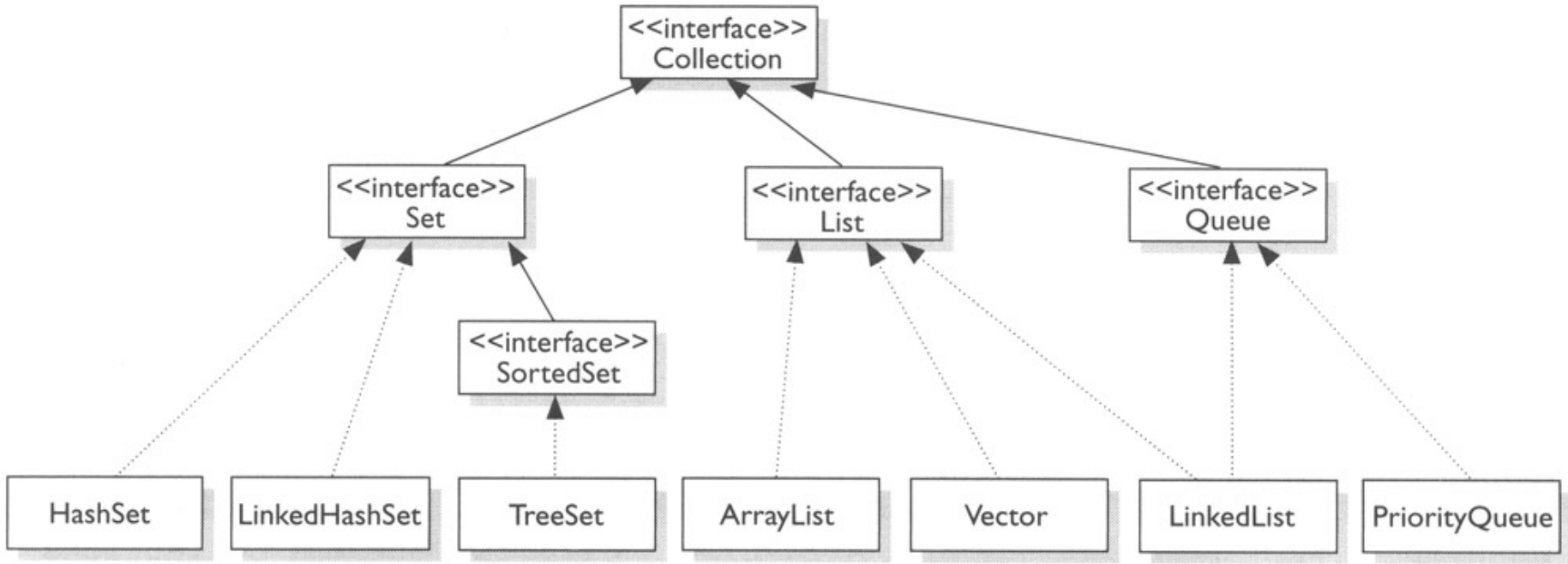
Evolving interfaces

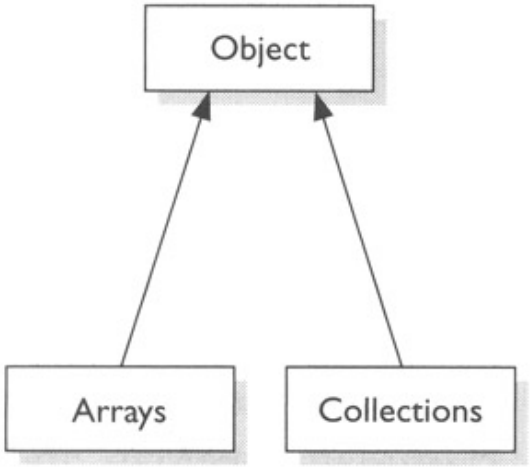
```
public class SalesRep extends Man implements Employee, Bonus
      {
   ...
}

public class Engineer extends Man implements Employee {
   ...
}
```

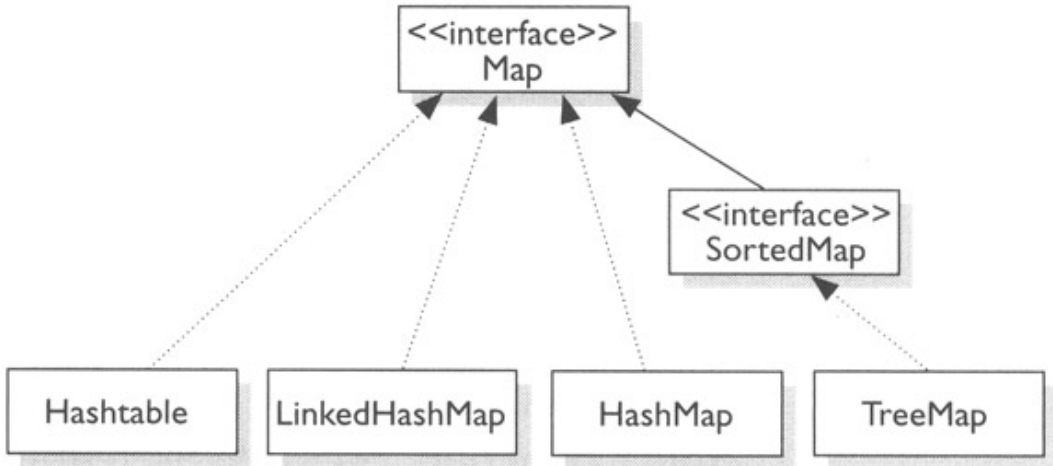
Evolving interfaces



Interfete celebre: o familie de colectii







Relatii intre interfete si intre interfete si clase

- o interfata poate extinde mai multe interfete (mostenire)
- o interfata poate fi implementata de mai multe clase
- o interfata nu poate fi instantiata
- o clasa poate implementa mai multe interfete

Clase Abstracte

Nevoia unor entitati intermediare

Payee:

- signContract este valabil pentru toti cei care sunt platiti de companie, deci poate fi implementat din start

Nevoia unor entitati intermediare

Payee:

- signContract este valabil pentru toti cei care sunt platiti de companie, deci poate fi implementat din start
- calcMoney este variabil in functie de cei care extind Payee (Employee, Bonus) deci poate fi declarat abstract si lasat spre implementare acestora

Definition

Clasa abstracta este o clasa declarata ca abstracta prin utilizarea cuvantului cheie "abstract". Desi nu este obligatoriu, acest lucru are sens sa se intample daca cel putin una din metodele sale este abstracta (adica declarata fara implementare). Reciproca insa este obligatorie: daca o clasa are cel putin o metoda abstracta, toata clasa trebuie marcata ca abstracta.

Ce poate contine o clasa abstracta

- orice contine o clasa obisnuita, doar ca este marcata ca abstracta (de fapt o clasa obisnuita poate fi marcata ca abstracta fara vreun motiv anume)
- elegant ar fi ca cel putin una din metodele unei clase abstracte sa fie abstracta (adica semnatura sa aiba cuvantul cheie abstract si sa nu aiba implementare)

Specificatorii de acces se pastreaza ca la o clasa obisnuita.

De ce sa folosim o clasa abstracta?

Clasa abstracta ofera o cale de mijloc intre clase si interfete, oferind posibilitatea de a crea clase incomplete, care vin cu un contract pentru a fi complet implementate de clasele ce le mostenesc:

- cand definim intr-un mod corect ceea ce au in comun clasele derivate

De ce sa folosim o clasa abstracta?

Clasa abstracta ofera o cale de mijloc intre clase si interfete, oferind posibilitatea de a crea clase incomplete, care vin cu un contract pentru a fi complet implementate de clasele ce le mostenesc:

- cand definim intr-un mod corect ceea ce au in comun clasele derivate
- cand dorim sa oferim o implementare partiala unei clase, ca sa nu duplicam cod (desi din Java 8 acest lucru se poate face si cu interfete si metode default)

Famous example: AbstractList

- AbstractList reference

Relatii intre clase abstracte, interfete si clase

- o clasa poate extinde o clasa abstracta daca ii implementeaza toate metodele abstracte
- daca o clasa extinde o clasa abstracta si nu ii implementeaza toate metodele abstracte, atunci, la randul sau, va fi marcata ca abstracta

Interfete vs Clase Abstracte

	Interfata	Clasa Abstracta
Se poate instantia	nu	nu
Metode	cu sau fara implementare	cu sau fara implementare
Membrii	static final	de orice fel
Specificatori acces	public (automat)	public, private, protected, default
Relatia cu alte clase	o clasa poate implementa oricate interfete	o clasa poate extinde o singura clasa abstracta

Cand sa folosim Interfete si cand Clase Abstracte?

Use abstract classes when:

- sharing code among several closely related classes, or
- classes that extend the abstract class have many common methods or fields, or require access modifiers other than public, or
- non-static or non-final fields are needed (methods that can access and modify the state of the object to which they belong)

Cand sa folosim Interfete si cand Clase Abstracte?

Use abstract classes when:

- sharing code among several closely related classes, or
- classes that extend the abstract class have many common methods or fields, or require access modifiers other than public, or
- non-static or non-final fields are needed (methods that can access and modify the state of the object to which they belong)

Use interfaces when:

- unrelated classes would implement the interface, or
- specifying the behavior of a particular data type, but not concerned about who implements its behavior, or
- multiple inheritance of type is needed

Pentru mai multe detalii ▸ Tutorialul despre Clase si Metode Abstracte

Mostenira multipla

Multiple inheritance

Definition

Mostenirea multipla este o functionalitate oferita de unele limbaje de programare prin care o clasa derivata preia caracteristicile mai multor clase parinte.

Multiple inheritance

Definition

Mostenirea multipla este o functionalitate oferita de unele limbaje de programare prin care o clasa derivata preia caracteristicile mai multor clase parinte.

In Java:

- o clasa poate extinde o singura alta clasa
- o clasa poate implementa mai multe interfete

Multiple inheritance

Definition

Mostenirea multipla este o functionalitate oferita de unele limbaje de programare prin care o clasa derivata preia caracteristicile mai multor clase parinte.

In Java:

- o clasa poate extinde o singura alta clasa
- o clasa poate implementa mai multe interfete

Din Java 8:

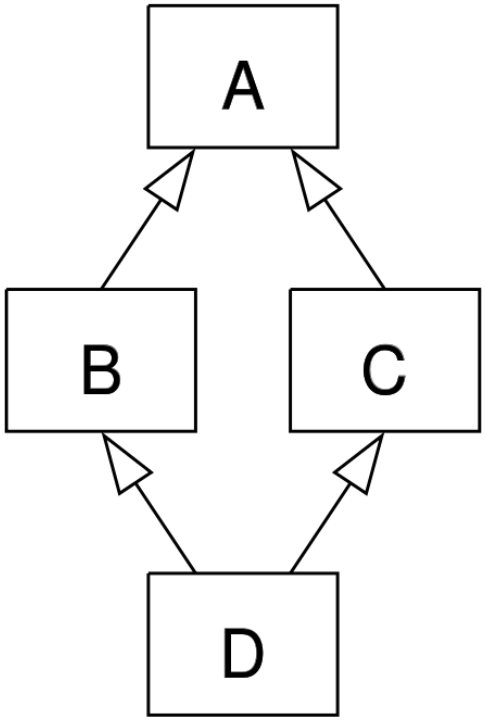
- o interfata poate contine metode default
- astfel, se poate implementa Mostenirea Multipla printr-o clasa care implementeaza mai multe interfete cu metode default

Problema diamant

Problema diamant ("diamond problem", aka "deadly diamond of death")

Problema diamant

Problema diamant ("diamond problem", aka "deadly diamond of death") se refera la ambiguitatea care apare atunci cand doua clase B si C mostenesc o clasa A, iar o alta clasa D mosteneste atat B, cat si C.



Daca exista o metoda in A, pe care o suprascriu atat B, cat si C, dar D nu o suprascrie, atunci D pe care din cele doua variante o va mosteni?

Problema diamant in Java 8 și inheritance prin interfaces

Programatorului îi este solicitat să rezolve ambiguitatea apelând B.super.m(...) sau C.super.m(...)

▸ What about the diamond problem? on lambdafaqs.org

Tipuri de mostenire multipla

- of State: not a problem, interface do not have fields
- of Implementation: default methods
- of Type: using an interface as a type

▸ Multiple Inheritance of State, Implementation, and Type

Mostenire multipla: interfete vs clase

Care este diferenta dintre:

- mostenirea multipla in sensul clasic (oferita de limbaje precum C++) in care o clasa poate mostenii mai multe clase
- mostenirea multipla obtinuta prin workaround-ul din Java8+ (o clasa poate implementa mai multe interfete, iar acestea pot mosteni o singura interfata)

Mostenire multipla: interfete vs clase

Care este diferenta dintre:

- mostenirea multipla in sensul clasic (oferita de limbaje precum C++) in care o clasa poate mostenii mai multe clase
- mostenirea multipla obtinuta prin workaround-ul din Java8+ (o clasa poate implementa mai multe interfete, iar acestea pot mosteni o singura interfata)

In mostenirea multipla din Java

- se mostenesc implementari, nu si stare (campuri)
- nu se mostenesc constructori

▸ Multiple Inheritance of State, Implementation, and Type

Factory

Description

Abstract Factory provides an interface for creating families of related or dependent objects without specifying their concrete classes.
Factory Method defines an interface for creating an object, but lets subclasses decide which class to instantiate.

Description

Abstract Factory provides an interface for creating families of related or dependent objects without specifying their concrete classes.
Factory Method defines an interface for creating an object, but lets subclasses decide which class to instantiate.

AbstractFactory classes are often implemented with Factory Methods, but they can also be implemented using Prototype. A concrete factory is often a Singleton.

Description

	AbstractFactory	Builder
what?	instances of related classes	complex object, step by step
object available	immediately	as a last step

Problem that the pattern solves

- vrem sa generam instante de clase inrudite (mostenesc o interfata comuna sau extind o clasa abstracta comuna)

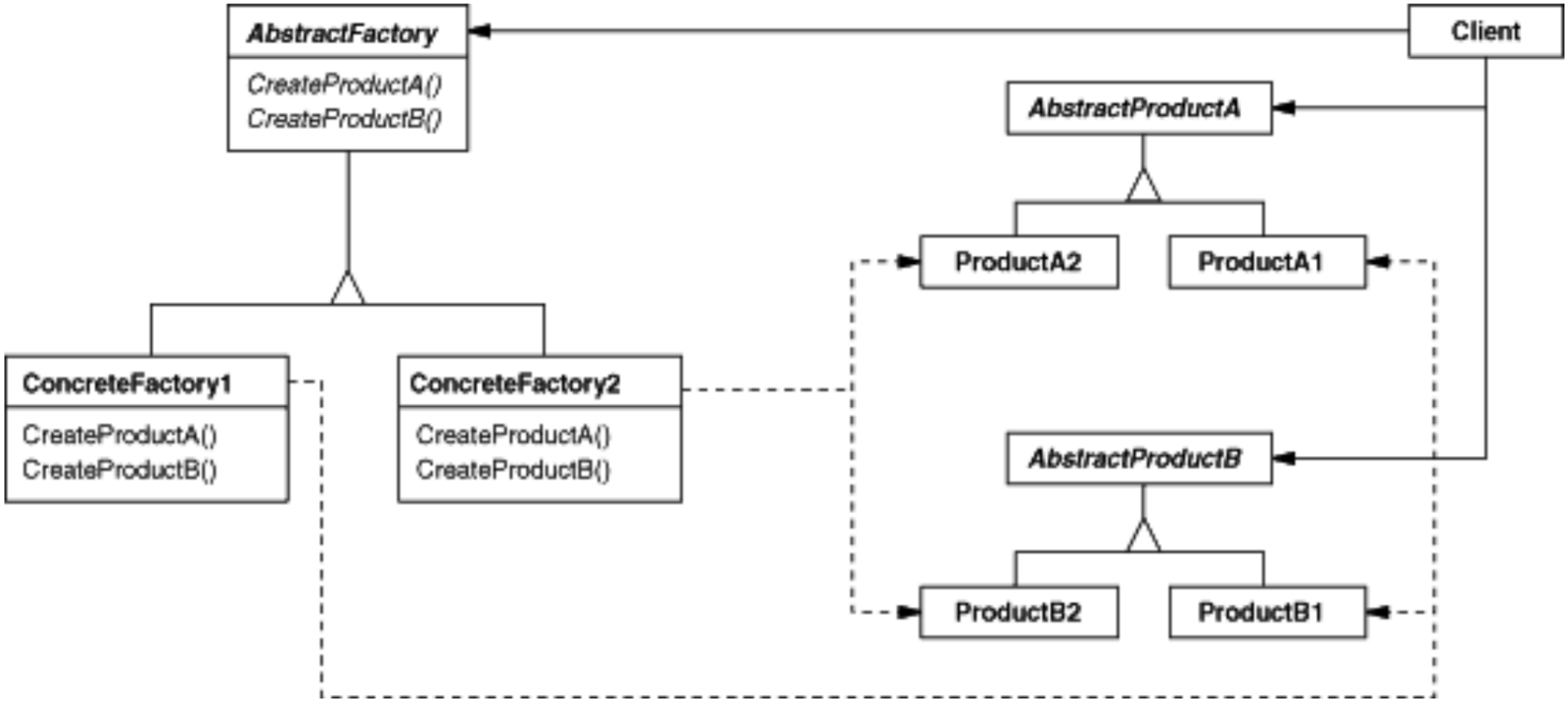
Problem that the pattern solves

- vrem sa generam instante de clase inrudite (mostenesc o interfata comuna sau extind o clasa abstracta comuna)
- clasa care solicita instantele nu are nevoie sa stie toate implementarile posibile, ci doar caracteristicile obiectului ce trebuie implementat

Problem that the pattern solves

- vrem sa generam instante de clase inrudite (mostenesc o interfata comuna sau extind o clasa abstracta comuna)
- clasa care solicita instantele nu are nevoie sa stie toate implementarile posibile, ci doar caracteristicile obiectului ce trebuie implementat
- vrem sa constrangem clientul sa foloseasca obiectele inrudite impreuna sau sa ii ascundem implementarile

Solution Structure



* E. Gamma, R. Helm, R. Johnson, J. Vlissides "Design Patterns: Elements of Reusable Object-Oriented Software"

- Products like soups, salads, main dishes etc.
- Factories like ItalianRestaurant, AsianRestaurant etc.

Solution Details

- de cele mai multe ori este suficienta o singura ConcreteFactory (dar sunt situatii in care putem avea mai multe)

Solution Details

- de cele mai multe ori este suficienta o singura ConcreteFactory (dar sunt situatii in care putem avea mai multe)
- crearea concreta a obiectelor apartine ConcreteFactory

Implementation Examples

Urmariti acest ▸ exemplu de Factory in C#

Exercitii

- Cautati care sunt clasele Factory in ▸ Java API
- Urmariti ▸ folosirea StringBuilder si sesizati diferentele dintre Factory si Builder
- Ar trebui sa puteti intelege tot ce scrie in ▸ acest articol
- Ce e prea mult strica: ▸ articol foarte interesant de la unul dintre fondatorii StackOverflow

Consequences

- izoleaza clientul de implementarea concreta a claselor

Consequences

- izoleaza clientul de implementarea concreta a claselor
- se poate schimba familia de produse usor (prin inlocuirea cu un alt Factory), dar adaugarea unui nou tip de produs este mai dificila din perspectiva clientului

Consequences

- izoleaza clientul de implementarea concreta a claselor
- se poate schimba familia de produse usor (prin inlocuirea cu un alt Factory), dar adaugarea unui nou tip de produs este mai dificila din perspectiva clientului
- promoveaza consistenta intre produse

Reading material

- Lab 5: Clase abstracte si interfete
- difference between an Interface and an Abstract class?
- 10 Abstract Class and Interface Interview Questions Answers in Java
- Why I hate frameworks, Benji Smith