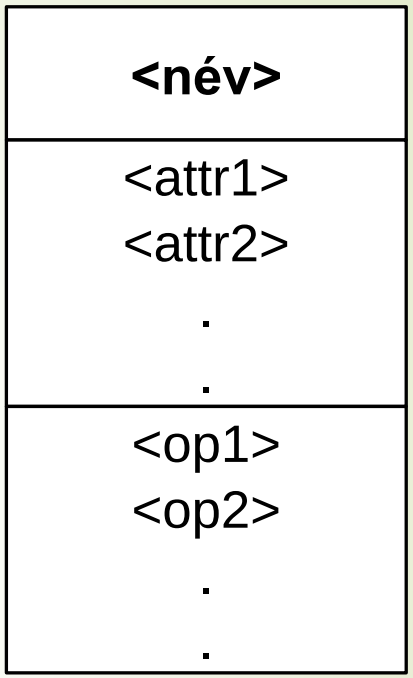
Programozási technológia

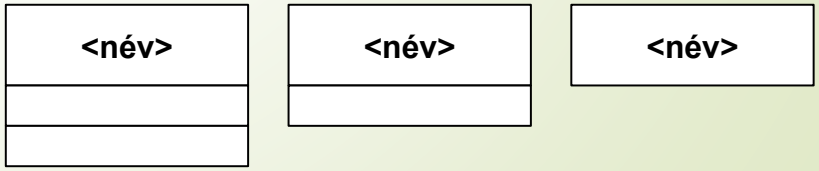
UML emlékeztető, Öröklődés

Dr. Szendrei Rudolf
ELTE Informatikai Kar
2020.

UML – Osztályok jelölése

- A diagramokban az osztály jelölésénél a nevét, az attribútumok nevét és a műveletek absztrakt formáját tüntetjük fel
- Az osztály neve félkövér betűkkel szedett
- Absztrakt osztály esetén az osztálynév félkövér dőlt betűkkel szedett
- Egyszerűsített jelölések:

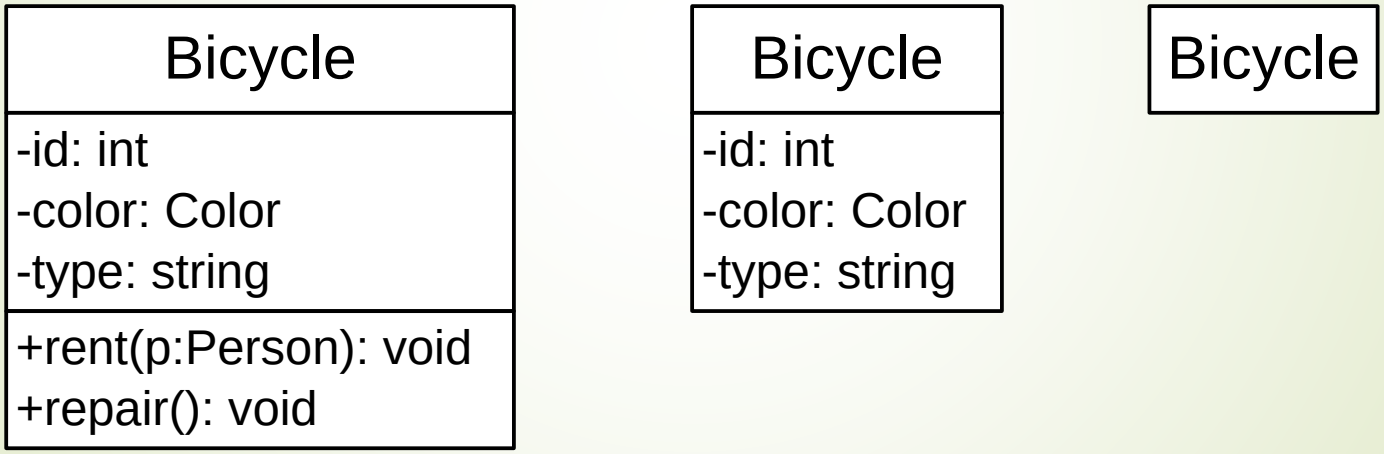




UML – Osztálydiagram példa

Példa: kerékpárok osztálya

- Ismert a kerékpárok színe, típusa, azonosítója
- Lehetséges műveletek: kölcsönzés, javítás



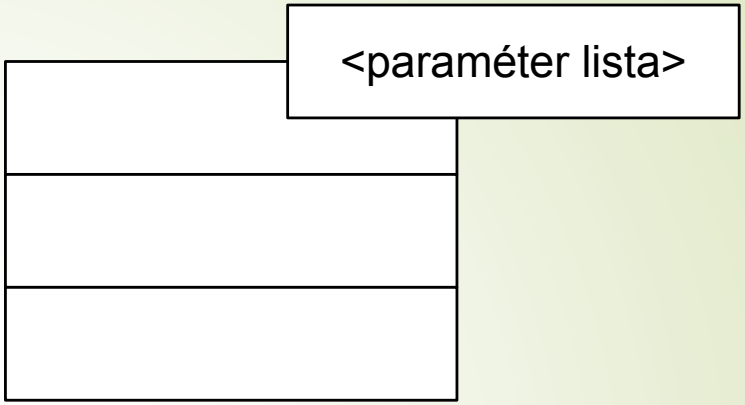
UML – Osztálydiagram példa

Kerékpár osztály Java-ban

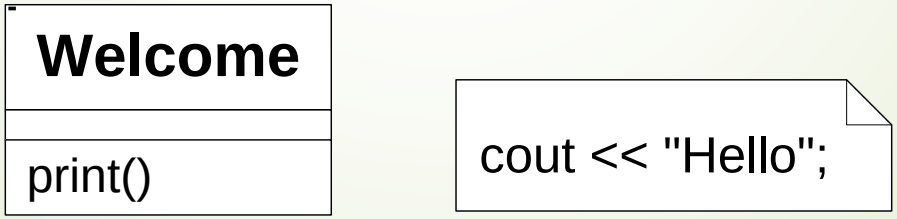
```
public class Bicycle {
  private int    id;
  private Color  color;
  private String type;


  public Bicycle(int id, Color color, String type) {...}


  public void rent(Person p){...}
  public void repair(){...}
}
```

UML – Osztályok jelölése

- Sablon osztály jelölése

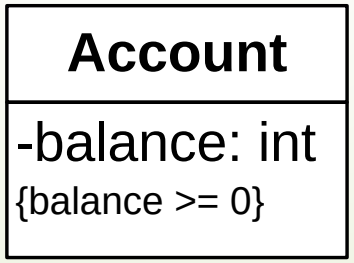


- Annotáció: szemantikus kiegészítés, mely az implementációs kérdéskörbe tartozó elemek jelölésére szolgál



UML – Megszorítások jelölése

- Az attribútumok lehetséges értékeire megszorításokat tehetünk
- Az attribútum mellé, kapcsos zárójelek közé írható a megszorítást kifejező feltétel
- Nem csak attribútumokra tehető megszorítás, UML diagramokban máshol is szerepelhetnek, ám minden esetben kapcsos zárójelek között kell megadni
- Példa: számlák esetén az egyenleg nem lehet negatív



UML – Osztálydiagram definíciója

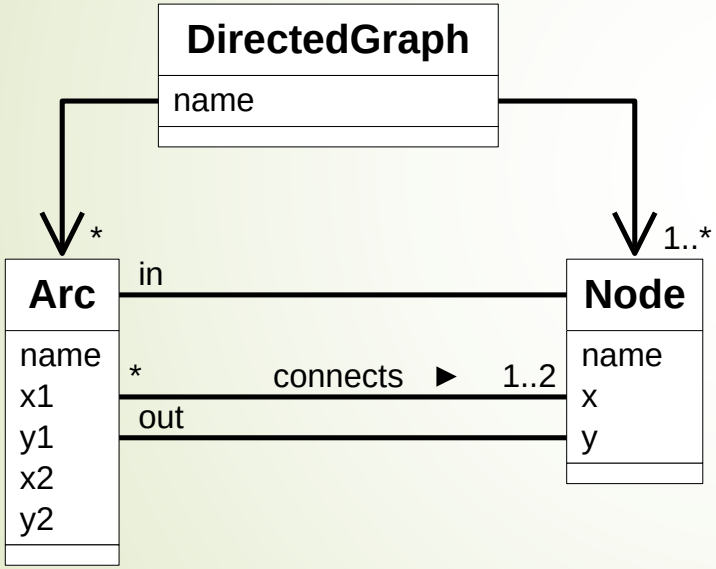
- Az osztálydiagram a problématérben a megoldás szerkezetét leíró összefüggő gráf, amelynek
  - csomópontjaihoz az osztályokat,
  - éleihez pedig az osztályok közötti relációkat rendeljük
- Osztályok között az alábbi relációk állhatnak fenn:
  - asszociáció
  - függőség
  - aggregáció
  - kompozíció
- öröklődés
- Megjegyzés: az öröklődés osztályok közötti kapcsolat, a többi a résztvevő osztályok objektumait kapcsolja össze

Asszociáció

- Két vagy több osztály objektumainak valamilyen relációval történő *összekapcsolása*
- Lehet reflexív, azaz azonos osztályú objektumok összekapcsolása is lehetséges
- Az asszociációhoz társulhat annak neve, *azonosítója*
- Lehet iránya: az aktív objektumtól a passzív objektum felé mutat
- Az összekapcsolt objektumoknak lehet multiplicitása, szerepe, és az összekapcsoláshoz minősítő is társulhat
- Navigálhatóság is megadható, amellyel kifejezhető, hogy az osztályok objektumai ismerik-e egymást
- A navigálhatóság elhagyása esetén kölcsönös elérhetőséget tételezünk fel

Asszociáció jelölése

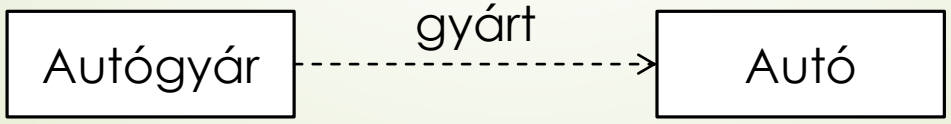
- Egyirányú navigálhatóság: asszociáció végén lévő nyíl
- Szereppel ellátott társítás: in, out
- Multiplicitás: *, 1..*, 1..2




```
class DirectedGraph{
  private String name;
  private List<Arc> arcs;
  private List<Node> nodes;
}

class Node{
  private String name;
  private List<Arc> inArcs;
  private List<Arc> outArcs;
}

class Arc{
  private String name;
  private Node inNode;
  private Node outNode;
}
```

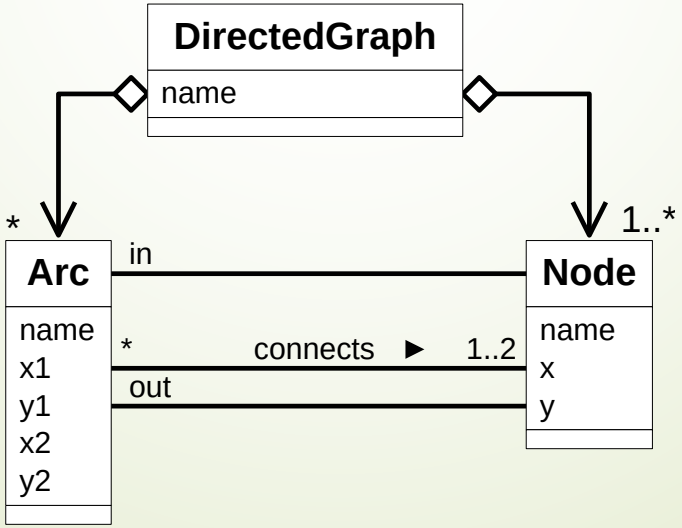
Függőség

- Egy osztály függ egy másiktól, ha a független osztályunk paraméterként vagy lokális változóként megjelenik egy, a függő osztály metódusánál
- Az asszociációhoz képest fontos eltérés, hogy ott a függő osztály attribútumaként jelenik meg a független osztálynak a példánya
- Függőségnél a kapcsolat néha annyira gyenge, hogy a függőséget generáló osztályhoz tagváltozó egyáltalán nem is jelenik meg a függő osztályban



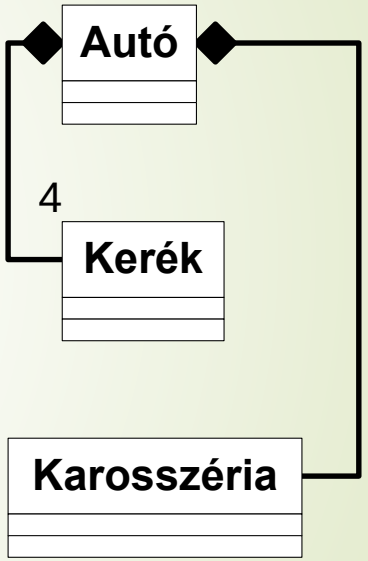
Aggregáció

- Speciális asszociáció
- Az általános asszociációnál erősebb kapcsolat, pl.:
  - Egész és annak részei,
  - Felépítmény és annak komponensei
- Azt fejezi ki, hogy az egyik osztály objektumai részét képezik egy másik osztály objektumainak
- Az aggregáció tranzitív, antiszimmetrikus, de nem lehet reflexív



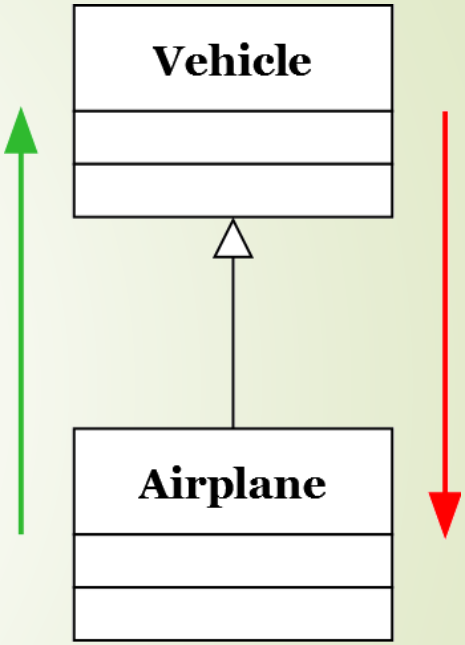
Kompozíció

- Speciális aggregáció
- Azt fejezi ki, hogy az egyik osztály objektumai a másik osztály objektumait fizikailag tartalmazzák
- A kompozíciós kapcsolat és az attribútum jellegű kapcsolat jelentése ugyanaz, csupán a diagramokban jelenik meg másképp
- Egy komponens objektum legfeljebb egy gazdaobjektumhoz tartozhat
- Egy gazdaobjektumnak tetszőleges számú komponense lehet
- A gazdaobjektum és annak komponensei azonos életciklusúak, azaz egyszerre jönnek létre, és egyszerre szűnnek meg



Általánosítás és specializáció

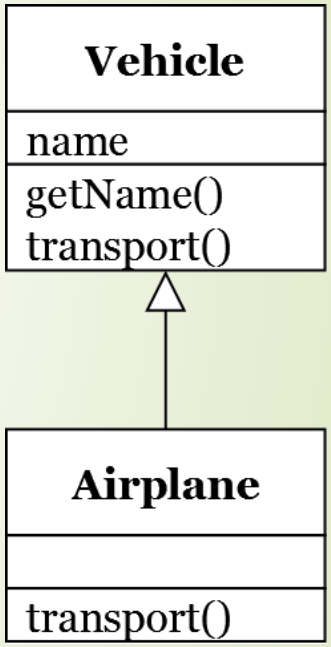
- Két osztály között állhat fenn
  - Általános osztály (ősosztály, superclass)
  - Általános tulajdonságokkal rendelkezik
  - Absztrakt metódusai is lehetnek
  - Speciális osztály
    (származtatott osztály, alosztály, subclass)
  - Speciálisabb tulajdonságokkal rendelkezik
  - Átveszi az általános osztály tulajdonságait, azokat kiegészítheti, átfogalmazhatja



- Példa: a repülő egyfajta jármű („is a kind of” reláció)

Általánosítás és specializáció

- Az általános osztály a szokásos módon rendelkezni fog a felsorolt attribútumokkal és műveletekkel.
- A speciális osztály megörökli az általános osztály minden attribútumát és operációját, ezeket nem kell újra feltüntetni a diagramon.
- Csak azoknak a tulajdonságoknak kell szerepelni a diagramon, amelyek csak a speciális osztályra vonatkoznak, és a megörököltek közül azok, amelyek valamilyen változáson mennek keresztül a speciális osztályban (például felüldefiniáltunk egy műveletet).



Általánosítás és specializáció

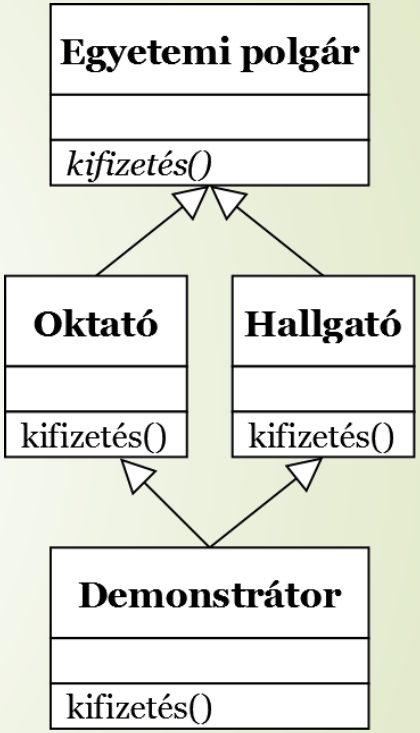
- A specializáció megvalósítása származtatással történik.

```
class Vehicle {...}
class Airplane extends Vehicle {...}
```

- A származtatás nem szimmetrikus, nem lehet reflexív.
- Új osztályok létrehozásának egy módja, mellyel absztrakt és konkrét osztályok egyaránt létrejöhetnek.
- A specializáció lehet többszörös, ekkor egy általánosításból több származott osztály jön létre.
- Az általánosítás is lehet többszörös, amikor a származtatás több általánosítással történik.
  - Java-ban a többszörös általánosítás csak úgy lehetséges, ha a több általános osztály közül legfeljebb az egyik konkrét osztály, a többi pedig legfeljebb interfész.

Általánosítás és specializáció

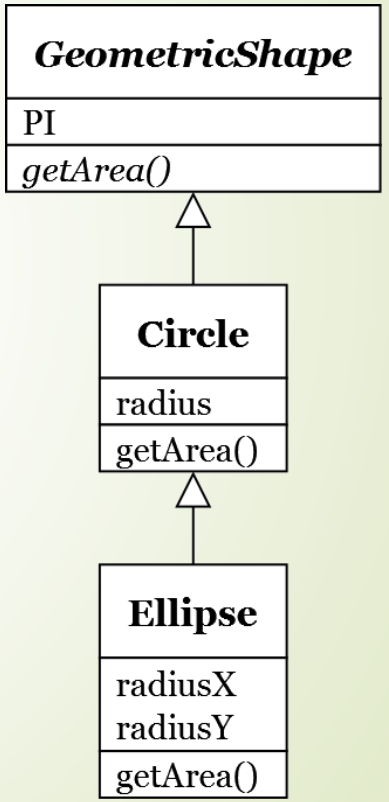
- Egy általános osztályból több speciális osztály is származtatható, és
- egy speciális osztály több általános osztály tulajdonságaival is rendelkezhet.
- Utóbbi esetben csak megfelelő korlátozó feltételekkel lehetséges az átvett, megörökölt információk összekeveredés nélküli kezelése.
- További specializációkkal az osztályok hierarchikus szerkezete hozható létre.



Öröklődés (származtatás)

Bármely megörökölt függvényt felül lehet definiálni (kivéve...), de csak akkor számít felüldefiniálásnak, ha a függvény szignatúrája pontosan ugyanaz, mint az ősosztályban.

```
public abstract class GeometricShape{
    protected final double PI = Math.PI;
    public abstract double getArea();
}

public class Circle extends GeometricShape{
    protected double radius;
    @Override
    public double getArea(){ return radius * radius * PI; }
}

public class Ellipse extends Circle{
    private double radiusY, radiusX = radius;
    @Override
    public double getArea(){
        return radiusX * radiusY * PI; }
}
```



Öröklődés (származtatás)

- Absztrakt függvény
  - Bármely függvénynek hiányozhat a megvalósítása, azaz a függvénytörzse (kivéve, ha a függvény vagy az osztálya `final`). Ekkor a függvény absztrakt függvény lesz.
- Absztrakt osztály
  - Ha egy osztály tartalmaz absztrakt függvényt, akkor az osztálynak is absztraktnak kell lennie.
  - Ha egy származtatott osztálynak absztrakt őse van, akkor vagy meg kell valósítania minden megörökölt absztrakt függvényt, vagy pedig a származtatott osztálynak is absztraktnak kell lennie.
  - Absztrakt osztályból nem lehet objektumokat példányosítani.

Öröklődés (származtatás)

- Java-ban az ősosztályra a `super` kulcsszóval hivatkozunk.
- A függvények megjelölhetőek a `final` kulcsszóval, ha azt akarjuk, hogy azokat ne lehessen felüldefiniálni (kivétel ez alól az absztrakt függvény).
- Egy teljes osztály is megjelölhető a `final` kulcsszóval (ha nem absztrakt osztály), ezzel megtiltjuk, hogy származtatni lehessen belőle.

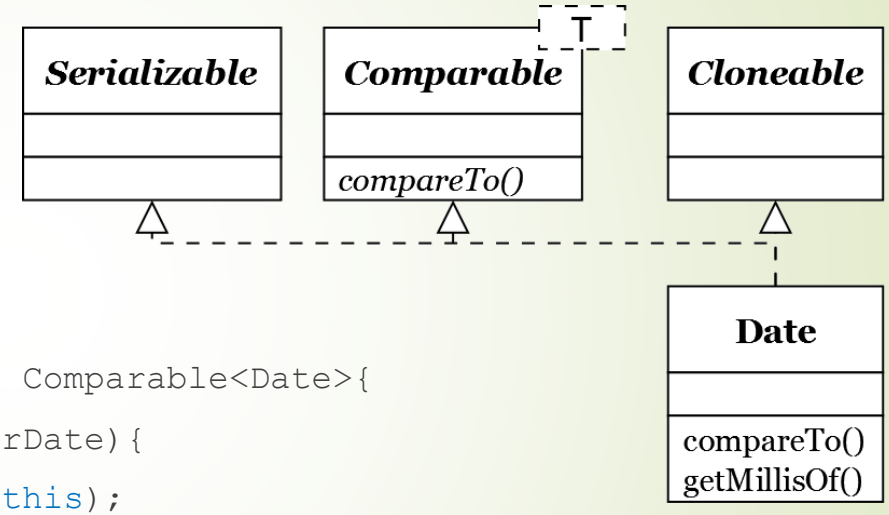
Öröklődés – Interfészek

- Az interfészek teljesen absztrakt osztályok, sőt akár üresek is lehetnek (ha csak egy típus megjelölést akarunk alkalmazni).
- Nem rendelkezhetnek megvalósítással, legfeljebb konstans attribútumokkal
  - egyik függvényének sem lehet függvénytörzse
- Ha teljesen absztrakt osztályból, interfészből származtatunk, akkor inkább megvalósításról (implementációról, realizációról) beszélünk.
- A Java interfészek bizonyos értelemben hasonlóak a C++ programok header fájljaihoz.
- Az osztályokhoz hasonlóan az interfészek is származtathatóak egymásból.
- Javaban csak interfészekből lehet több ősosztály (és legfeljebb egy konkrét osztályból).
- Az interfészeket konkrét osztályokkal valósítjuk meg.

Öröklődés – Interfészek (példa)

- Az, hogy a Comparable interfész egyben egy generikus típus is, az az öröklődéstől teljesen független.

```
public interface Comparable<T> {
  public int compareTo(T o);
}

public class Date implements
    java.io.Serializable, Cloneable, Comparable<Date>{
    public int compareTo(Date anotherDate){
        long thisTime = getMillisOf(this);
        long anotherTime = getMillisOf(anotherDate);
        return (thisTime<anotherTime ? -1 :
                (thisTime==anotherTime ? 0 : 1));
    }
}
```



Öröklődés

- Az absztrakt osztályok objektum példányosításkor helyben specializálhatóak (ekkor egy új, névtelen osztály fog létrejönni).

```
GeometicShape shape = new GeometicShape(){
    @Override
    public double getArea(){
        return 0.0;
    }
}
```

Öröklődés – Polimorfizmus

- Mivel az öröklődés egy ,,is a kind of..." reláció, ezért a speciálisabb objektumok behelyettesíthetőek az általánosabb objektumok helyére.
- Többalakúság (polimorfizmus): egy `Square` objektum felveheti a `Square` osztály tulajdonságait, de az általános `GeometricShape` osztály tulajdonságait is (mivel megörökli azokat).

```
List<GeometricShape> shapes = new ArrayList<>();
shapes.add(new Circle());
shapes.add(new Ellipse());
shapes.add(new Square());

for (GeometricShape geomShape : shapes) {
    System.out.println(geomShape.getArea());
}
```