VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

10 - OBJEKTINIS PROGRAMAVIMAS 3

Jaroslav Grablevski / Justina Balsė

Casting

```
class Animal {
    void makeNoise() {
        System.out.println("generic noise");
    }}
class Dog extends Animal {
    void makeNoise() {
        System.out.println("bark");
    }
    void playDead() {
        System.out.println("roll over");
    }}
class CastTest2 {
    public static void main(String[] args) {
        Animal[] a = { new Animal(), new Dog(), new Animal() };
        for (Animal animal : a) {
            animal.makeNoise();
            if (animal instanceof Dog) {
                animal.playDead(); // <- won't compile
                Dog d = (Dog) animal; // casting the reference variable
                d.playDead();         // sometimes called a downcast, because we're casting
                                      // down the inheritance tree to a more specific class
            }
        }
    }
}
```

Casting

Compiles but fails later:

```
class Animal {}
class Dog extends Animal {}
class DogTest {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Dog d = (Dog) animal; // compiles but fails later
    }
}
```

Will NOT compile :

```
public class Pzv {
    Animal animal = new Animal();
    Dog d = (Dog) animal;
    String s = (String) animal; // animal can't EVER be a String
}
```

instanceof

The **instanceof** operator tests if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface.

```
class Animal { }
class Dog extends Animal { }
```

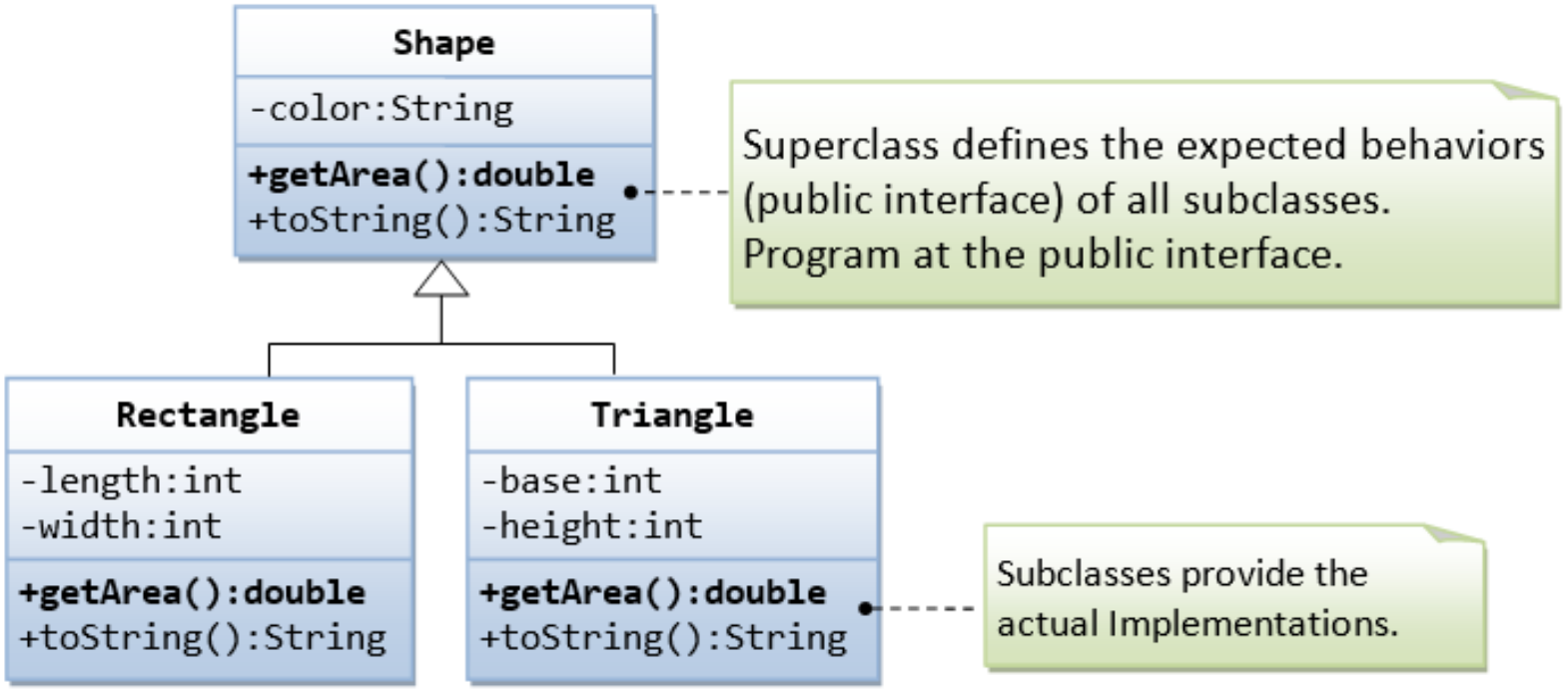
```
Dog dog = new Dog();
Animal animal = new Animal();
System.out.println(dog instanceof Dog);     //true
System.out.println(dog instanceof Animal); //true
System.out.println(dog instanceof Object); //true

System.out.println(animal instanceof Dog); //false
dog = null;
System.out.println(dog instanceof Dog);     //false
```

Polimorfizmas

- objektiniame programavime naudojama sąvoka, kai metodas gali būti vykdomas skirtingai, priklausomai nuo konkrečios klasės realizacijos, metodo kvietėjui nieko nežinant apie tokius skirtumus.
- “moku dirbti su vienu tipu, moku dirbti su visais tipais, kurie įgyvendina tą patį kontraktą”
  - Grįstas paveldėjimu ir įgyvendinimu*
  - Jei *Y extends X*, Y galime naudoti visur kur tikimes gauti X

Polimorfizmas



Polimorfizmas pvz.

```
public class Shape {
   // All shapes must have a method called getArea().
   public double getArea() {
      System.err.println("Shape unknown! Cannot compute area!");
      return 0;  // We need to return some value to compile the program.
   }
}
```

```
public class Rectangle extends Shape {
    private int length, width;
    public Rectangle(int length, int width) {..
    @Override // Override to provide the proper implementation
    public double getArea() {
        return length * width;
    }
}
```

```
public class Triangle extends Shape {
    //...
    public double getArea() {
        return 0.5 * base * height;
    }
    //..
```

Polimorfizmas pvz.

```
public class TestShape {
    public static void main(String[] args) {
        Shape s1 = new Rectangle(4, 5); // Upcast
        System.out.println("Area is " + s1.getArea()); // Runs Rectangle's getArea()

        Shape s2 = new Triangle(4, 5); // Upcast
        System.out.println("Area is " + s2.getArea()); // Runs Triangle's getArea()

    }
}
```

```
Area is 20.0
Area is 10.0
```

Polimorfizmas pvz. 2

```
public class Animal {
    public void eat() {
        System.out.println("Generic Animal Eating Generically");
    }
}
```

```
public class Horse extends Animal {
    public void eat() {
        System.out.println("Horse eating hay ");
    }

    public void eat(String s) {
        System.out.println("Horse eating " + s);
    }
}
```

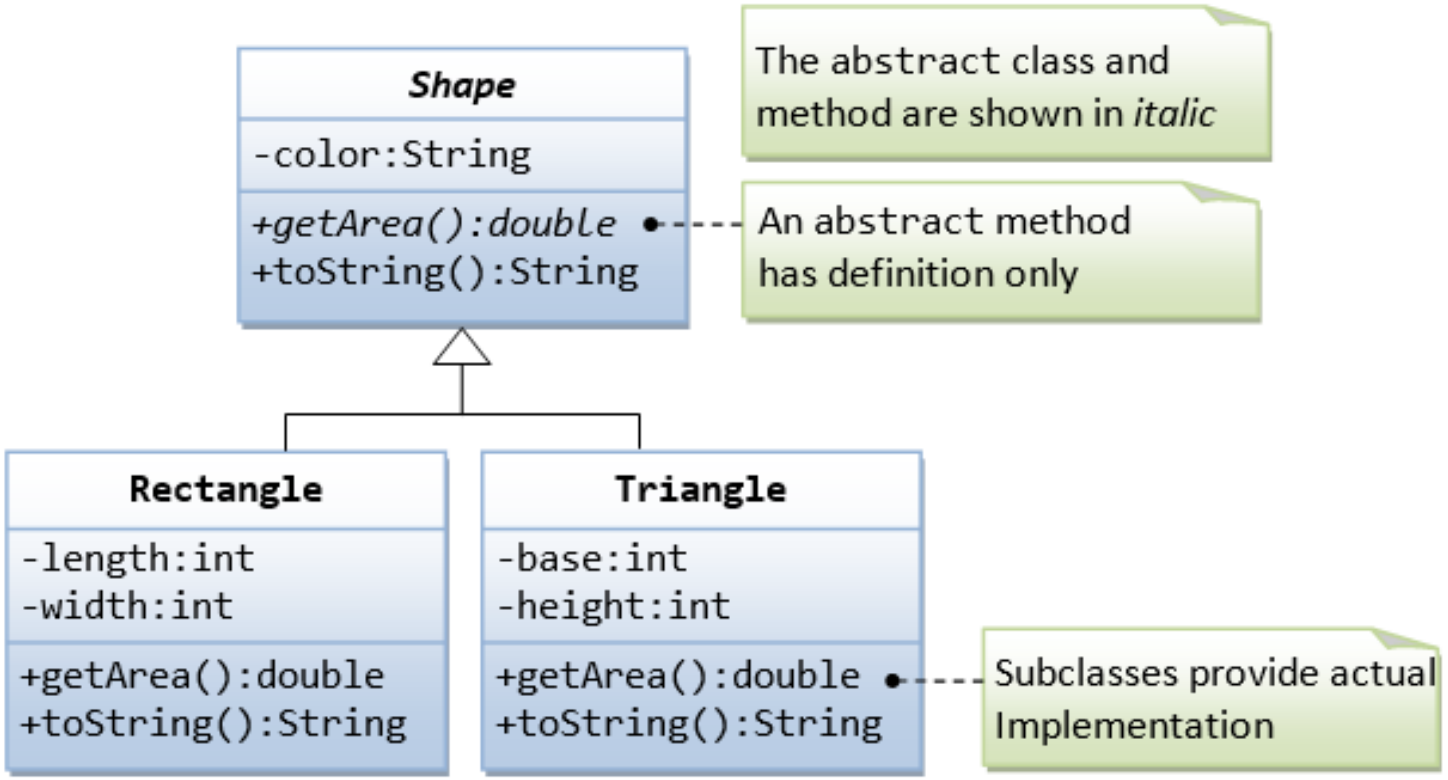
Polimorfizmas pvz. 2

Method Invocation Code	Result
`Animal a = new Animal();` `a.eat();`	`Generic Animal Eating Generically`
`Horse h = new Horse();` `h.eat();`	`Horse eating hay`
`Animal ah = new Horse();` `ah.eat();`	`Horse eating hay` Polymorphism works—the actual object type (`Horse`), not the reference type (`Animal`), is used to determine which `eat()` is called.
`Horse he = new Horse();` `he.eat("Apples");`	`Horse eating Apples` The overloaded `eat(String s)` method is invoked.
`Animal a2 = new Animal();` `a2.eat("treats");`	Compiler error! Compiler sees that the `Animal` class doesn't have an `eat()` method that takes a `String`.
`Animal ah2 = new Horse();` `ah2.eat("Carrots");`	Compiler error! Compiler still looks only at the reference and sees that `Animal` doesn't have an `eat()` method that takes a `String`. Compiler doesn't care that the actual object might be a `Horse` at runtime.

Abstrakti klasė

- Abstrakčios klasės turi būti pažymėtos raktažodžiu *abstract;*
- Negalima sukurti objektų iš abstrakčių klasių;
- Gali turėti nerealizuotų/abstrakčių metodų;
- Abstrakti klasė skirta paveldėjimui - paveldinti klasė turi realizuoti visus abstrakčius metodus arba pati būti abstrakčia;

Abstrakti klasė



Abstrakti klasė

```
abstract public class Shape {
    // Private member variable
    private String color;

    // All Shape subclasses must implement a method called getArea()
    abstract public double getArea();

    // Constructor
    public Shape(String color) {
        this.color = color;
    }

    @Override
    public String toString() {
        return "Shape [color=" + color + "]";
    }
}
```

```
Shape s3 = new Shape("green");    // Compilation Error!!
```

Interfeisas

- Tai yra visiškai abstrakti klasė, kurioje egzistuoja grupė susijusių metodų, kurie yra nerealizuoti*.
- Naudojama nusakyti kontraktą
  - “Aš esu …”
  - “Aš moku …”
- Klasė gali įgyvendinti daug interfeisų

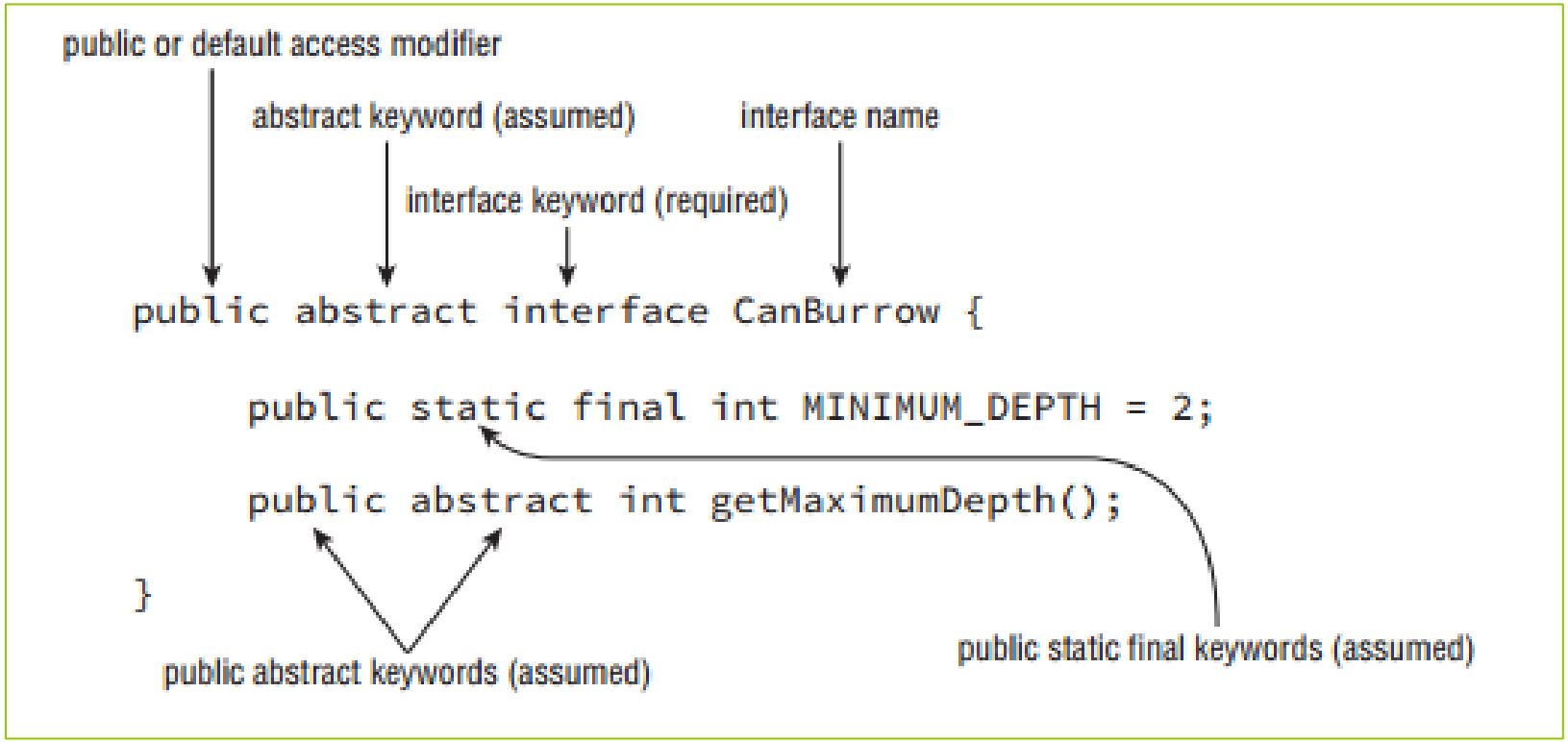
*Nuo Java 8 gali tureti realizuotus metodus (*default* ir *static*)

Interface vs Abstract class

	Java 7 and Earlier	Java 8 and Later
Abstract Classes	• Can have concrete methods and abstract methods • Can have static methods • Can have instance variables • Class can directly extend one	(Same as Java 7)
Interfaces	• Can only have abstract methods – no concrete methods • Cannot have static methods • Cannot have mutable instance variables • Class can implement any number	• Can have concrete (default) methods and abstract methods • Can have static methods • Cannot have mutable instance variables • Class can implement any number

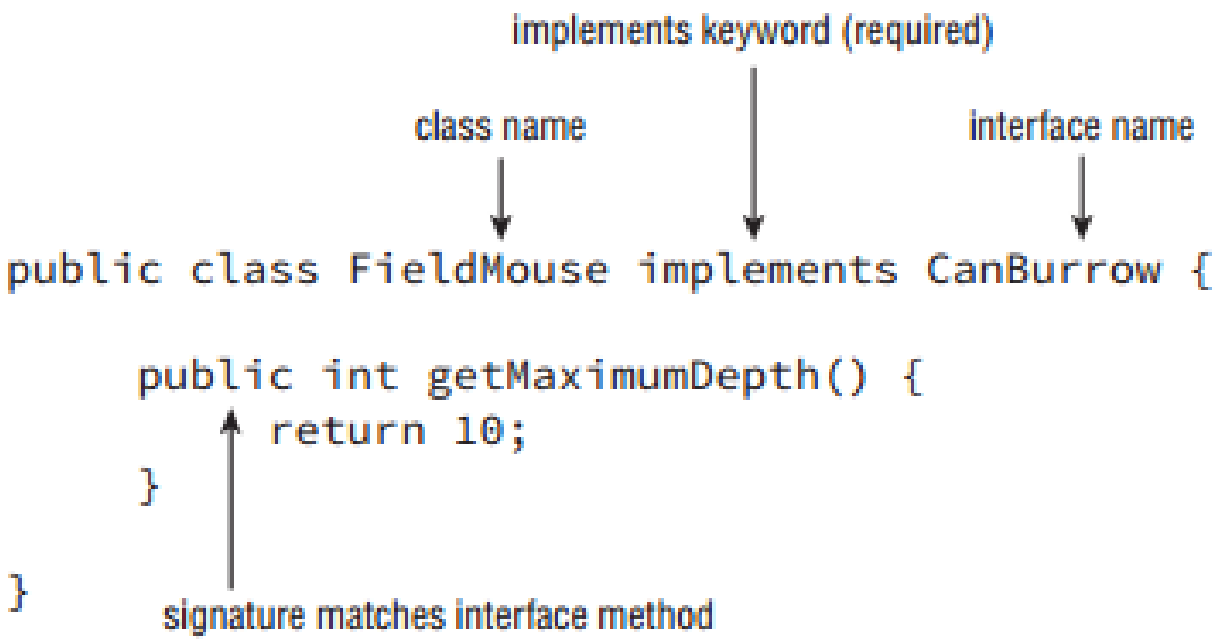
Interfeisas

- Defining an interface

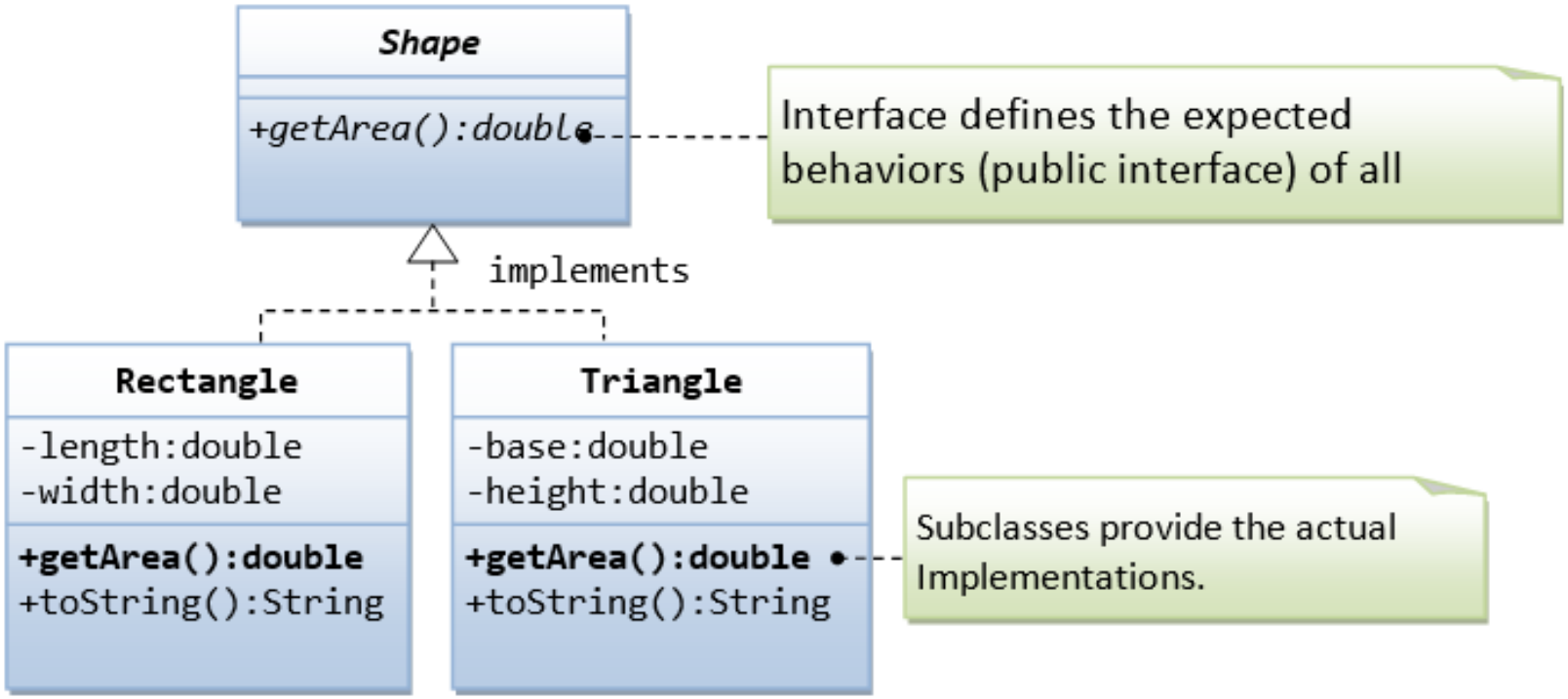


Interfeisas

- Implementing an interface



Interfeisas



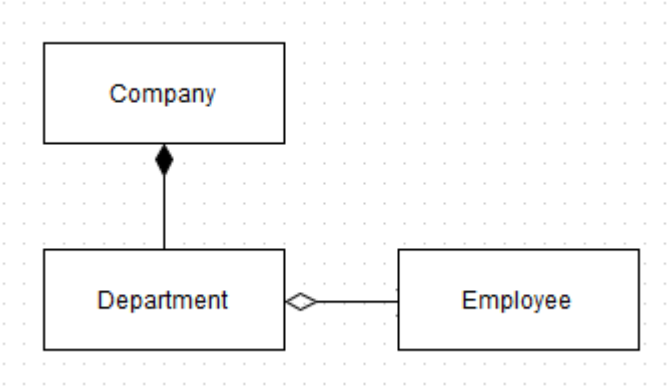
Interfeisas

```
public interface Shape {  // Use keyword "interface" instead of "class"
   // "public abstract" assumed
   double getArea();
}
```

```
public class Rectangle implements Shape {
   //...
```

Agregavimas ir kompozicija

- Agregavimo ryšys vadinamas „turi“(angl. **has-a**) ryšiu
- Agregavimas yra silpnesnis sudėtinio objekto ryšys
  - vienos klasės objektas yra kitos objekto dalis, tačiau sunaikinus vieną, kitas nebūtinai turi būti sunaikinamas
- Kompozicijos ryšys – stipresnis
  - sunaikinus vienos klasės objektą sunaikinamos ir jo sudedamosios dalys



Agregavimas