TECHIN

9 - OBJEKTINIS PROGRAMAVIMAS (II)

Jaroslav Grablevski

Turinys

- Paveldėjimas
- Metodų užklotis (overriding)
- equals()
- @Override
- super

Paveldėjimas (inheritance)

- Tai klasės gebėjimas paveldėti protėvių klasės duomenis (kintamuosius, laukus) ir metodus.
- Klasė, kuri yra išvesta iš kitos klasės vadinama poklase arba vaikine (angl. subclass, derived class, extended class arba child class).
- Klasė, iš kurios poklasė yra išvesta, vadinama superklase arba tėvine (angl. superclass, base class, parent class) .

Paveldėjimas

- Tai klasės gebėjimas paveldėti protėvių klasės duomenis (kintamuosius, laukus) ir metodus.
- Tėvinė klasė apibrėžia tam tikrus kintamuosius bei metodus, kuriuos vaikinė klasė paveldi.
- Vaikinė klasė gali pasipildyti savais kintamaisiais ir metodais
- Vaikinė klasė turi galimybę pakeisti tėvinių metodų veikimą juos perrašant (angl. override)

Paveldėjimas

- Java kalboje paveldėjimo ryšys deklaruojamas naudojant žodelį extends po kurio nurodoma klasė iš kurios yra paveldima

```
public class Car extends Vehicle {
    //....
}
```

Paveldėjimas

Programmer

name
address
phoneNumber
experience
programmingLanguages
writeCode()

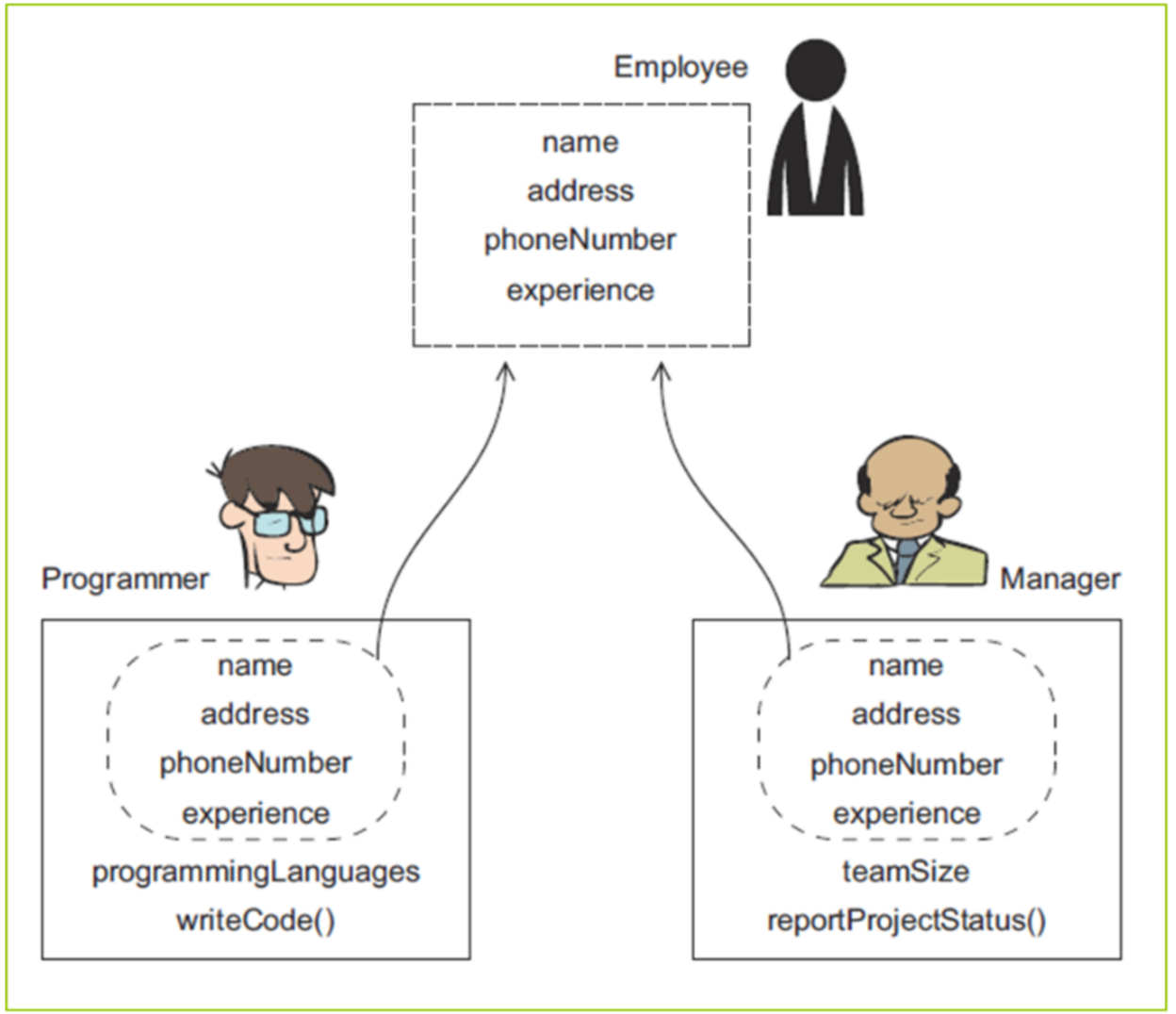
```
class Programmer {
    String name;
    String address;
    String phoneNumber;
    float experience;
    String[] programmingLanguages;
    void writeCode() {}
}
```

Manager

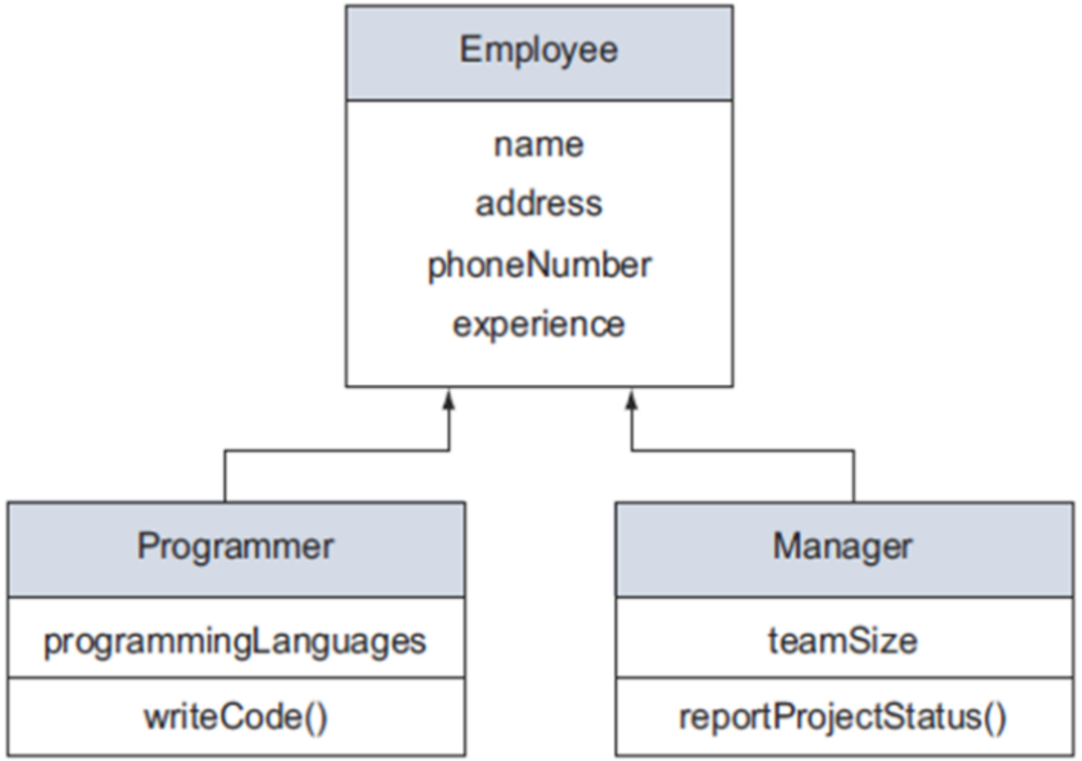
name
address
phoneNumber
experience
teamSize
reportProjectStatus()

```
class Manager {
    String name;
    String address;
    String phoneNumber;
    float experience;
    int teamSize;
    void reportProjectStatus() {}
}
```

Paveldėjimas

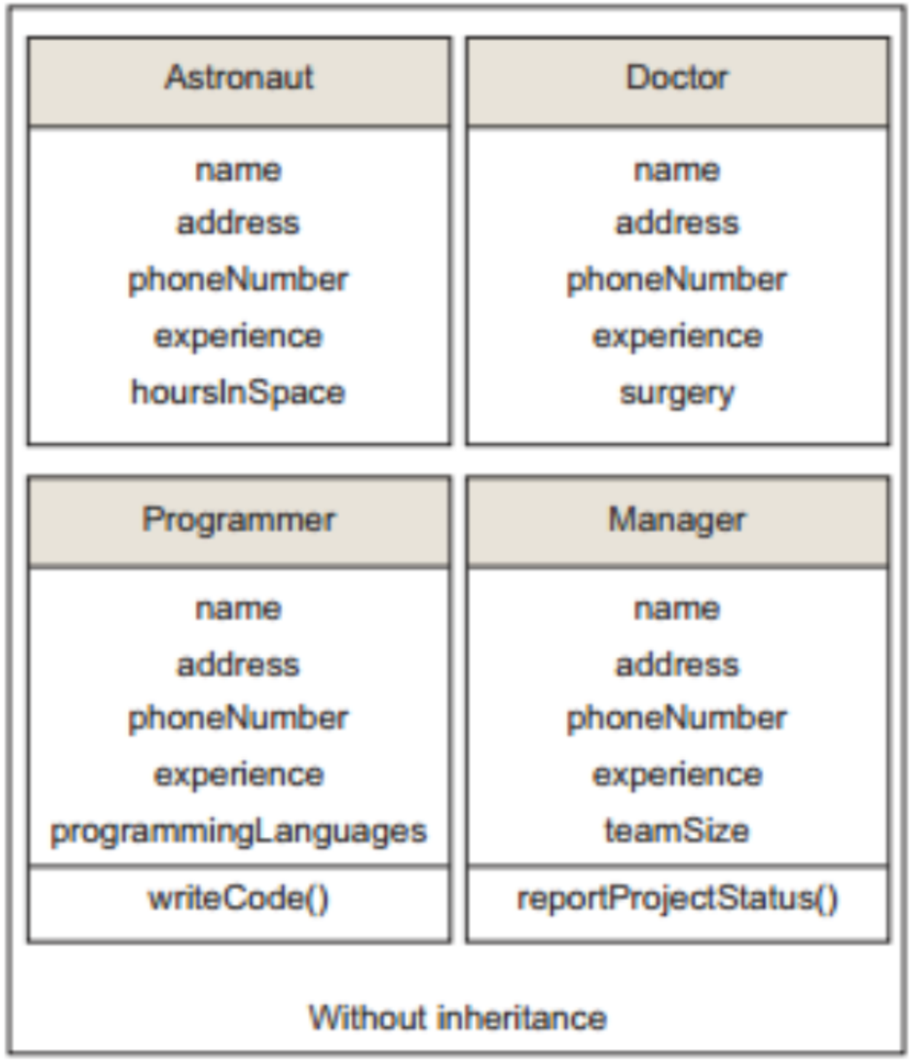


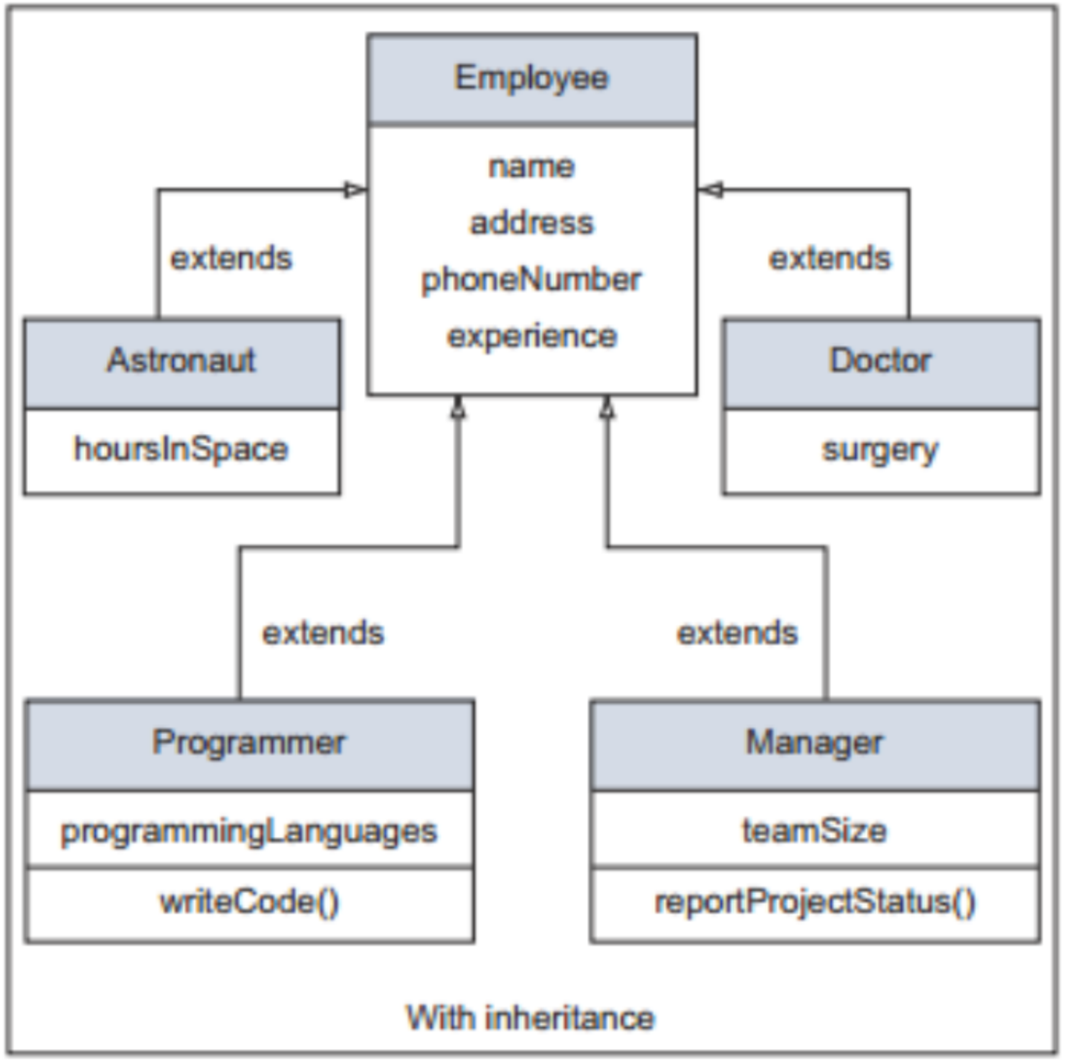
Paveldėjimas



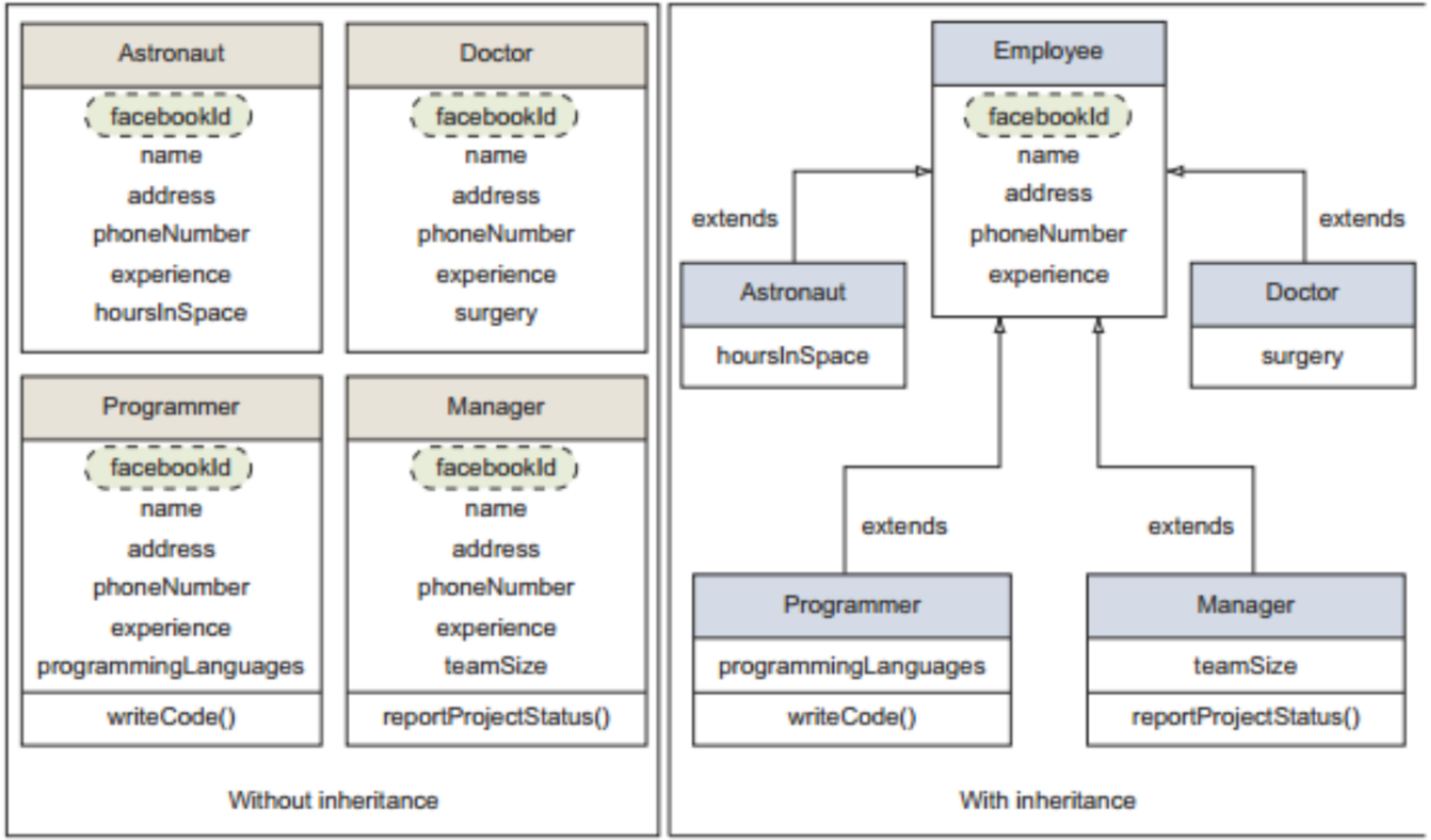

```
class Employee {
     String name;
     String address;
     String phoneNumber;
     float experience;
}
class Programmer extends Employee {
     String[] programmingLanguages;
     void writeCode() {}
}
class Manager extends Employee {
     int teamSize;
     void reportProjectStatus() {}
}
```

Paveldėjimas





Paveldėjimas



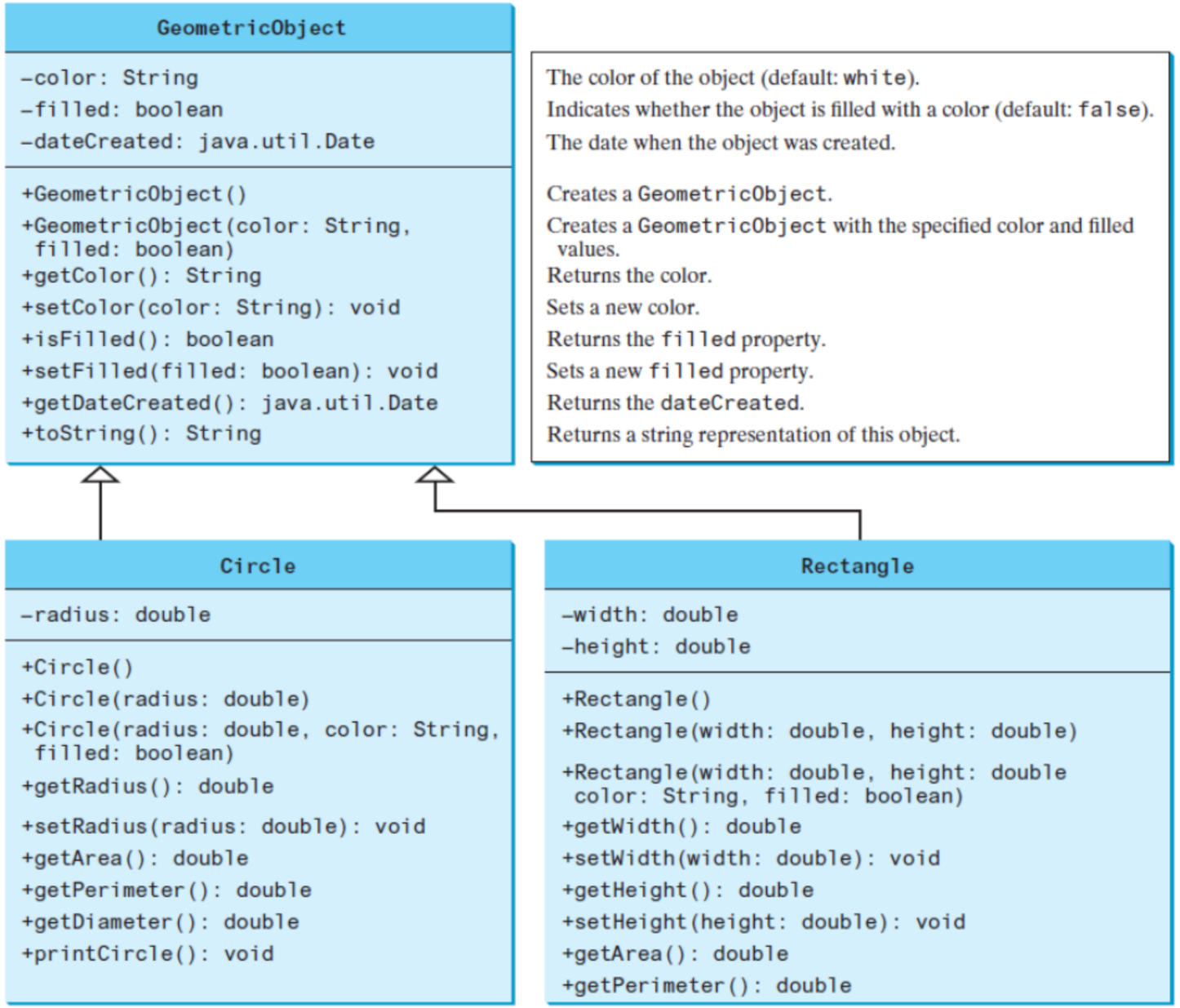
Paveldėjimas

Subclass → Superclass

```
public class Circle extends GeometricObject
```

Subclass → Superclass

```
public class Rectangle extends GeometricObject
```



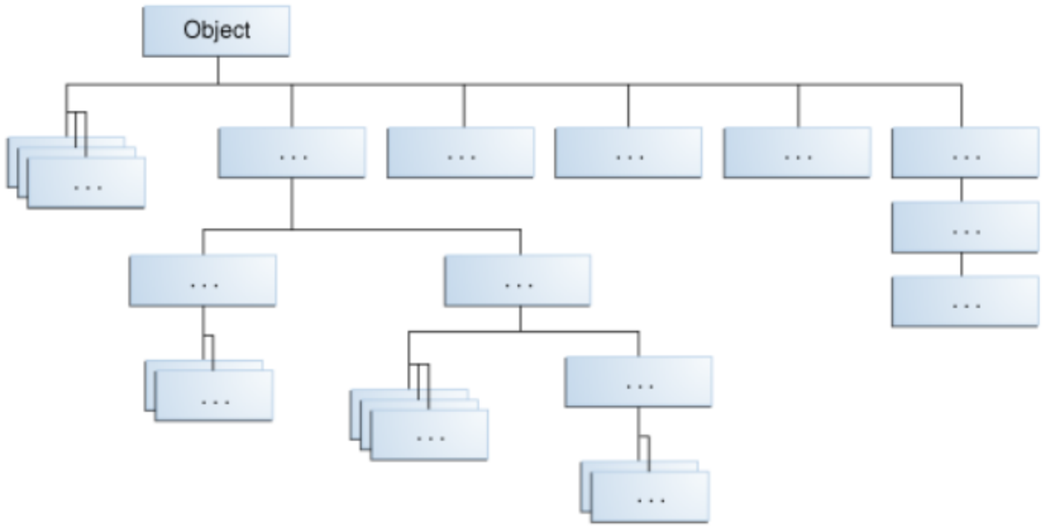
Paveldėjimas

- Klasė gali turėti tik vieną tiesioginę superklasę
- Visos Javos klasės yra kilusios iš java.lang.Object klasės ir automatiškai paveldi visus jos metodus

```
public class ClassName {
  ...
}
```

Equivalent

```
public class ClassName extends Object {
  ...
}
```



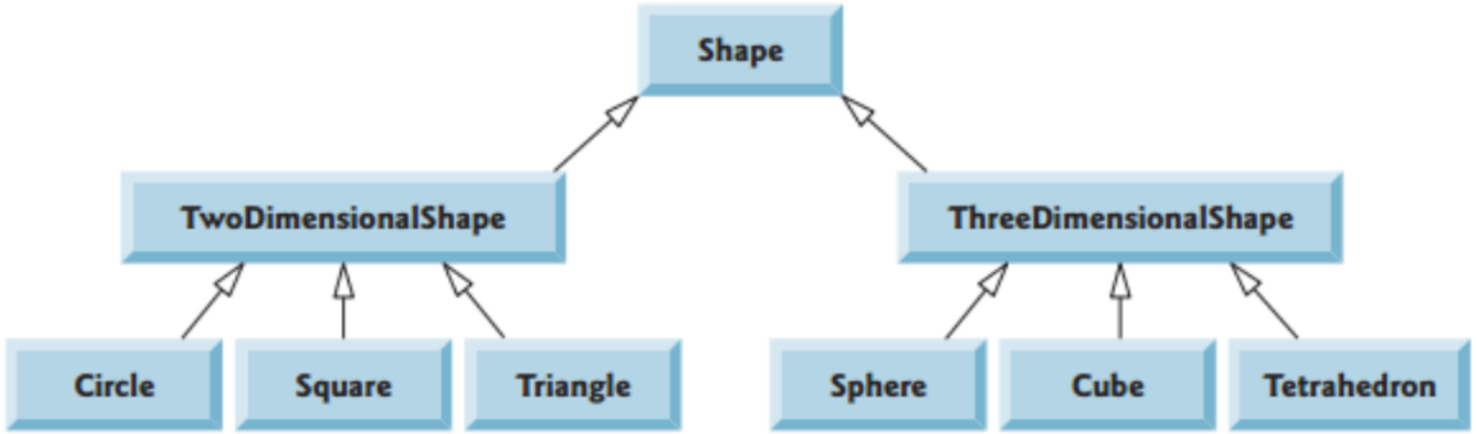
java.lang.Object class

- Object class contains behaviors that are inherited by all Java classes:
  - The toString method creates text value for the object.
  - The equals method indicates whether some other object is "equal to" this particular one.
  - The hashCode method generates int hash value for the object.
  - The clone method produces a replica of the object.
  - getClass returns an instance of Class , which has information about the runtime class
  - wait, notify, and notifyAll methods control threads.

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

is-a relationship

Superclass	Subclasses
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
BankAccount	CheckingAccount, SavingsAccount



Machine ir Car klasės | Pavyzdys (1)

```java
public class Machine {

    public void start() {
        System.out.println("Machine started.");
    }

    public void stop() {
        System.out.println("Machine stopped.");
    }
}
```

```java
public class Car extends Machine {

}
```

Main klasė | Pavyzdys (1.2)

```java
public class Main {

    public static void main(String[] args) {

        Machine m1 = new Machine();

        System.out.println("Machine class: ");
        m1.start();
        m1.stop();

        Car m2 = new Car();
        // Machine m2 = new Car();

        System.out.println("Car class: ");
        m2.start();
        m2.stop();

    }
}
```

```
Machine class:
Machine started.
Machine stopped.
Car class:
Machine started.
Machine stopped.
```

Papildome Car klasę | Pavyzdys (2)

- Vaikinė klasė papildoma jai trūkstamais metodais.

```
public class Car extends Machine {

    public void turnRight() {
        System.out.println("Car turned right.");
    }

    public void turnLeft() {
        System.out.println("Car turned left.");
    }

}
```

Main klasė | Pavyzdys (2.1)

```
public class Main {

    public static void main(String[] args) {

        Car m2 = new Car();

        System.out.println("Car class: ");

        m2.start();
        m2.stop();

        m2.turnLeft();
        m2.turnRight();

    }
}
```

```
Car class:
Machine started.
Machine stopped.
Car turned left.
Car turned right.
```

Metodų užklotis / Overriding (1)

- Vaiko klasėje paveldėtus Tėvo klasės metodus galima keisti.
- Turi sutapti:
  - metodų vardai
  - jų antraštės
  - metodų grąžinamos reikšmės tipai (gali būti labiau specifinis tipas)
- Priėjimo modifikatorius išvestinėje klasėje neturi padidinti metodo uždarumo laipsnio
- Užklojantysis metodas neturi mesti naujų checked exceptions
- Neužklojami
  - konstruktoriai
  - final tipo metodai
  - statiniai metodai (bet galima „paslėpti“ statiniu metodu)

Machine ir Car klasės | Pavyzdys (3)

```java
public class Machine {

    public void start() {
        System.out.println
        ("Machine started.");
    }

    public void stop() {
        System.out.println
        ("Machine stopped.");
    }
}
```

```java
public class Car extends Machine {

    @Override
    public void start() {
        System.out.println
        ("Car started.");
    }

    @Override
    public void stop() {
        System.out.println
        ("Car stopped.");
    }

    public void turnRight() {

    public void turnLeft() {

}
```

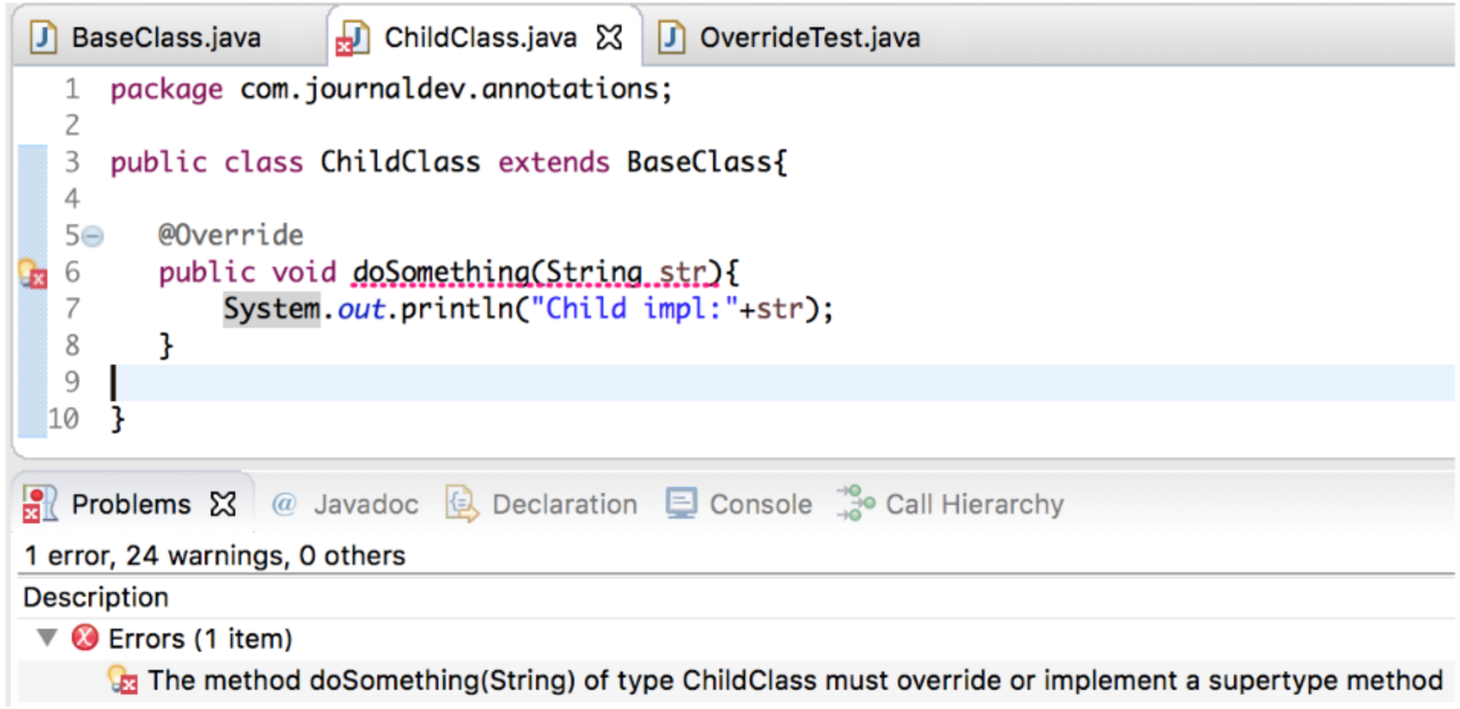
Main klasė | Pavyzdys (3.1)

```
public class Main {

    public static void main(String[] args) {

        Machine m1 = new Machine();

        System.out.println("Machine class: ");
        m1.start();
        m1.stop();

        Car m2 = new Car();

        System.out.println("Car class: ");
        m2.start();
        m2.stop();

    }
}
```

```
Machine class:
Machine started.
Machine stopped.
Car class:
Car started.
Car stopped.
```

@Override – anotacija

- Norint pabrėžti, kad metodas yra perrašytas galima nurodyti anotaciją @Override, nors tai nėra būtina
- Taip užtikrinama kontrolė, kad metodas perrašytas, o ne naujai sukurtas



equals()

```
//The default implementation of the equals
//method in the java.lang.Object class is:
public boolean equals(Object obj) {
    return this == obj;
}
```

```
Person p1 = new Person("John", "Smith", 31);
Person p2 = new Person("John", "Smith", 31);

System.out.println(p1.equals(p2)); // false
```

equals()

```java
@Override
public boolean equals(Object other) {
    /* Check this and other refer to the same object */
    if (this == other) {
        return true;
    }
    /* Check other is Person and not null */
    if (!(other instanceof Person)) {
        return false;
    }
    Person person = (Person) other;
    /* Compare all required fields */
    return age == person.age &&
            Objects.equals(firstName, person.firstName) &&
            Objects.equals(lastName, person.lastName);
}
```

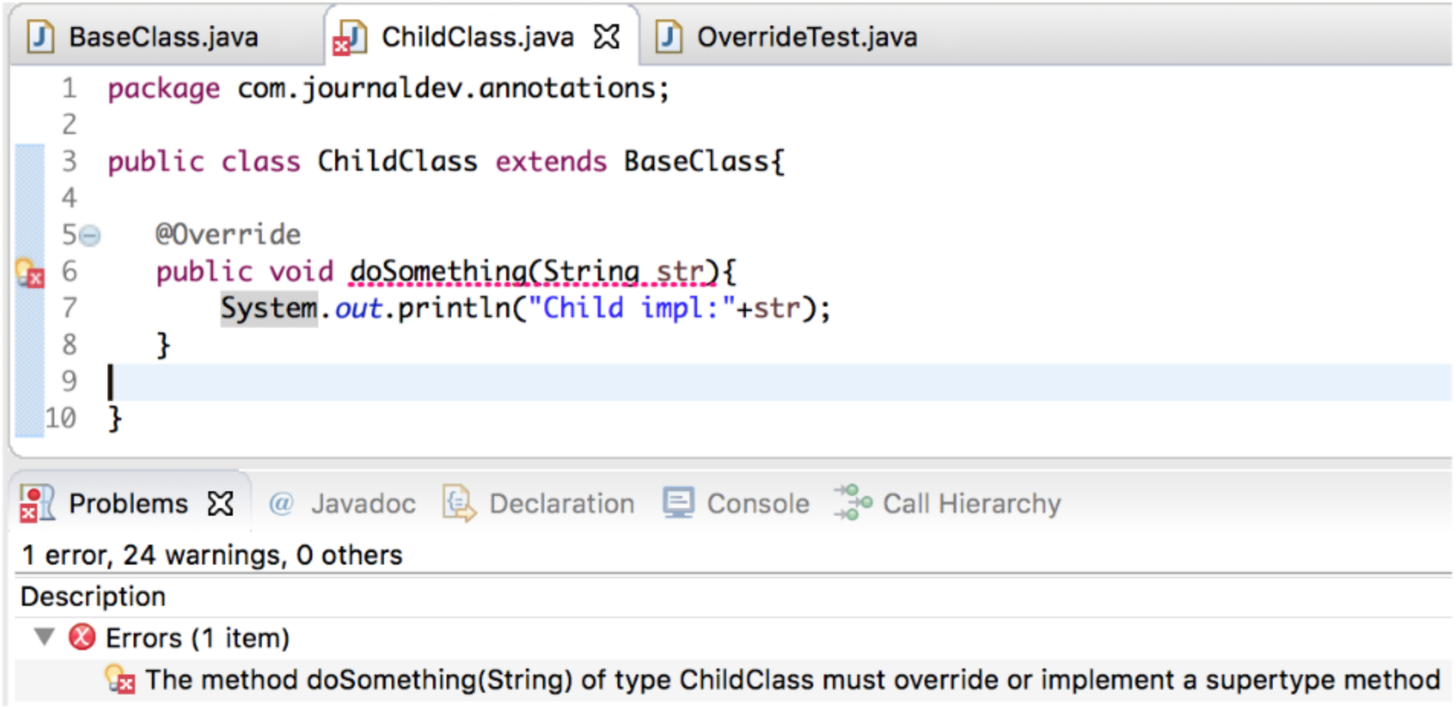
equals()

- Po *equals()* perrašymo

```
Person p1 = new Person("John", "Smith", 31); // a person
Person p2 = new Person("John", "Smith", 31); // the same person
Person p3 = new Person("Marry", "Smith", 30); // another person

System.out.println(p1.equals(p2)); // true
System.out.println(p2.equals(p3)); // false
System.out.println(p3.equals(p3)); // true (reflexivity)
```

@Override – anotacija

- Norint pabrėžti, kad metodas yra perrašytas galima nurodyti anotaciją @Override, nors tai nėra būtina
- Taip užtikrinama kontrolė, kad metodas perrašytas, o ne naujai sukurtas



super

- Perrašančio metodo viduje yra galimybė kviesti perrašomą metodą naudojant žodelį super.

```
public class Superclass {

    public void printMethod() {
        System.out.println("Printed in Superclass.");
    }
}
```

```
public class Subclass extends Superclass {

    // overrides printMethod in Superclass
    public void printMethod() {
        super.printMethod();
        System.out.println("Printed in Subclass");
    }
    public static void main(String[] args) {
        Subclass s = new Subclass();
        s.printMethod();
    }
}
```

```
Printed in Superclass.
Printed in Subclass
```

super in methods

```
public class Circle extends GeometricObject {
    private double radius;
    // Other methods are omitted

    // Override the toString method defined in the superclass
    public String toString() {
        return super.toString() + "\nradius is " + radius;
    }
}
```

super

- Norint iškviesti tam tikrą tėvinės klasės konstruktorių, naudojamas žodelis super(...).
- Šis sakinys turi būti pats pirmasis sakinys konstruktoriuje.
- Jei toks sakinys nėra nurodytas, tuomet java kompiliatorius automatiškai jį priskiria - kviečiamas konstruktorius be argumentų.

```
public ClassName() {
  // some statements
}
```

Equivalent

```
public ClassName() {
  super();
  // some statements
}
```

```
public ClassName(parameters) {
  // some statements
}
```

Equivalent

```
public ClassName(parameters) {
  super();
  // some statements
}
```

super

```
class Employee {
    String name;
    String address;

    Employee(String name, String address) {
        this.name = name;
        this.address = address;
    }
}

class Programmer extends Employee {
    String progLanguage;

    Programmer(String name, String address, String progLang) {
        super(name, address);
        this.progLanguage = progLang;
    }
}
```

super

```
//Implicit super constructor Fruit() is undefined for default
//constructor. Must define an explicit constructor
public class Apple extends Fruit {
}

class Fruit {
    public Fruit(String name) {
        System.out.println("Fruit's constructor is invoked");
    }
}
```

super

```
public class C extends B{
    public static void main(String[] args) {
        new C();
    }
    public C() {
        System.out.println("C's constructor called");
    }
}
class B extends A{
    public B() {
        System.out.println("B's constructor called");
    }
}
class A {
    public A() {
        System.out.println("A's constructor called");
    }
}
```

super

```
public class C extends B{
    public static void main(String[] args) {
        new C();
    }
    public C() {
        super();
        System.out.println("C's constructor called");
    }
}
class B extends A{
    public B() {
        super();
        System.out.println("B's constructor called");
    }
}
class A {
    public A() {
        super();
        System.out.println("A's constructor called");
    }
}
```

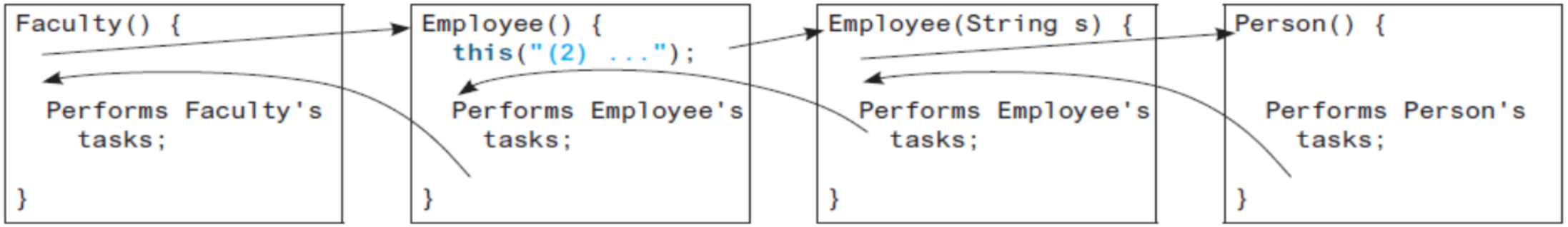
```
A's constructor called
B's constructor called
C's constructor called
```

super ir this

```
public class Faculty extends Employee {
    public static void main(String[] args) {
        new Faculty();
    }
    public Faculty() {
        System.out.println("Performs Faculty's tasks");
    }
}
class Employee extends Person {
    public Employee() {
        this("Employee's overloaded constructor");
        System.out.println("Performs Employee's tasks ");
    }
    public Employee(String s) {
        System.out.println(s);
    }
}
class Person {
    public Person() {
        System.out.println("Performs Person's tasks");
    }
}
```

super ir this

```
Performs Person's tasks
Employee's overloaded constructor
Performs Employee's tasks
Performs Faculty's tasks
```



Final Classes and Methods

- The final keyword can be used to limit class extensibility
- Class cannot extend a class that is marked with the final keyword.
- The subclass cannot override a superclass method that is marked within the final keyword.

```
public class Product {
  public final void processPayment() {
    // method can not be overridden by subclasses
  }
}
```

```
public class Drink extends Product {
  public void processPayment() {}
}
```

```
public final class Food extends Product {
  // class can not be extended
}
```

```
public class JunkFood extends Food { }
```