Sąsajos ir paveldėjimai

Code Academy

Programuok savo ateitį!

Code Academy
Programuok savo ateitį!

Paveldėjimai

Java kalboje klasės gali būti kilusios iš kitų klasių, taip paveldint jų laukus ir metodus

Code Academy
Programuok savo ateitį!

Paveldėjimai

Klasė, kilusi iš kitos klasės, yra vadinama vaikine klase. Klasė, iš kurios kyla kitos klasės, vadinama super klase arba tėvine klase

Code Academy
Programuok savo ateitį!

Paveldėjimai

Java klasės gali turėti tik vieną super klasę. Jei klasė neturi nurodytos super klasės, ji paveldi klasę Object pagal nutylėjimą

Code Academy
Programuok savo ateitį!

Paveldėjimai

Vaikinė klasė paveldi narius (laukus, metodus, vidines klases) iš tėvinės klasės. Vaikinės klasės konstruktoriai privalo kviesti super klasės konstruktorių, nes jie nėra paveldimi. Kiekvieno konstruktoriaus pirma eilutė turi kviesti tėvinės klasės konstruktorių

Code Academy
Programuok savo ateitį!

Paveldėjimai

Paveldimumas taikomas siekiant naudoti laukus arba metodus, egzistuojančius kitose klasėse, neperrašant jų pačiam.

Code Academy
Programuok savo ateitį!

Paveldėjimai

Vaikinės klasės paveldi visus **public** ir ***protected*** tėvinės klasės narius. Taip pat, jei vaikinė klasė yra tame pačiame kataloge (***package***) kaip ir tėvinė klasė, ji paveldi visus ***package-private*** narius

Code Academy
Programuok savo ateitį!

Klasės laukų pasiekiamumas

Modifikatorius	Tos pačios klasės nariai	To paties paketo klasės	To paties paketo vaikinės klasės nariai	Vaikinės klasės nariai	Visos kitos klasės
private	Taip	Ne	Ne	Ne	Ne
	Taip	Taip	Taip	Ne	Ne
protected	Taip	Taip	Taip	Taip	Ne
public	Taip	Taip	Taip	Taip	Taip

Code Academy
Programuok savo ateitį!

Casting

Vaikinės klasės gali turėti naują, tik joms būdingą funkcionalumą, tačiau jos taip pat turi tėvinės klasės savybes. Tai reiškia, kad vaikinė klasė gali būti panaudota ten, kur reikalinga jos tėvinė klasė

Code Academy
Programuok savo ateitį!

Casting

Klasė TemperatureSensor kuri paveldi klasę **Sensor** gali būti priskirta **TemperatureSensor**, **Sensor**, **Object** tipo kintamiesiems.
Visų šių teiginių sintaksė yra teisinga:

```
TemperatureSensor temperatureSensor = new TemperatureSensor();
Sensor sensor = temperatureSensor;
Object object = temperatureSensor;
```

Code Academy
Programuok savo ateitį!

Casting

Klasės tipo keitimas į jos tėvinių objektų tipą vadinamas ***upcast‘inimu***.
Upcast‘inus objektą į tėvinės klasės tipą, vaikinei klasei būdingi metodai tampa nematomi.
Upcast‘inimas vyksta automatiškai.

Code Academy
Programuok savo ateitį!

Casting

Tėvinius objektus galima ***cast‘inti*** į vaikinės klasės tipus, tai vadinama ***downcasting***.
Downcast‘inimas nevyksta automatiškai, tam reikia specifiškai nurodyti ***cast‘inimo*** operatorių

Code Academy
Programuok savo ateitį!

Casting

Downcast'inimas - tiesiog nurodymas kompiliatoriui, kad tu žinai, koks iš tiesų yra objekto tipas veikimo (*runtime*) metu.

```
TemperatureSensor temperatureSensor = new TemperatureSensor();
Sensor sensor = temperatureSensor;
TemperatureSensor otherTemperatureSensor = (TemperatureSensor) sensor;
```

Code Academy
Programuok savo ateitį!

Casting

Downcast'inti objektus reikėtų atsargiai, nes „neatspėjus" objekto tipo programa pateikia ***ClassCastException***

```
Sensor sensor = new HumiditySensor();
TemperatureSensor incorrectSensor = (TemperatureSensor) sensor;
```

Code Academy
Programuok savo ateitį!

Casting

Įšvengti **ClassCastException** padeda operatorius ***instanceOf***, naudojamas nusakyti objekto tipui

```
Sensor sensor = new HumiditySensor()
System.out.println(sensor instanceof HumiditySensor);
System.out.println(sensor instanceof TemperatureSensor);
```

Code Academy
Programuok savo ateitį!

Casting

Java leidžia ***downcast'inti*** objektus tik į tokius tipus, kurie paveldi tėvinę klasę

```
String text = "simple text";
Sensor sensor = (Sensor) text;
```

Code Academy
Programuok savo ateitį!

Metodų perrašymas

Vaikinės klasės gali perrašyti ne ***final*** tėvinės klasės metodus. Tai leidžia pritaikyti klasės veikimą pagal savo poreikius.
Situacija, kai objekto tipas nulemia metodo veikimą, vadinama ***polimorfizmu***

Code Academy
Programuok savo ateitį!

Metodų perrašymo pavyzdys

```
public abstract class Sensor {
    public abstract String purpose();
}
public class TemperatureSensor extends Sensor {
    @Override
    public String purpose() {
        return "measures temperature";
    }
}
public class HumiditySensor extends Sensor {
    @Override
    public String purpose() {
        return "measures humidity";
    }
}

public static void main(String[] args) {
    Sensor temperatureSensor = new TemperatureSensor();
    Sensor humiditySensor = new HumiditySensor();
    System.out.println(temperatureSensor.purpose());
    System.out.println(humiditySensor.purpose());
}
```

Code Academy
Programuok savo ateitį!

Metodų perrašymas

Metodų perrašymas ypatingas tuo, kad visada kviečiamas perrašytas metodas - net ***upcastin‘us*** klasės tipą į tėvinį, bus kviečiamas ne tėvinis metodas, o perrašyta metodo versija

Code Academy
Programuok savo ateitį!

Metodų perrašymo pavyzdys

```
public class HumiditySensor extends Sensor {
    @Override
    public String purpose() {
        return "measures humidity";
    }
}

public class AdvancedHumiditySensor extends HumiditySensor {
    @Override
    public String purpose() {
        return "precisely measures humidity";
    }
}

public static void main(String[] args) {
   AdvancedHumiditySensor advancedSensor = new AdvancedHumiditySensor();
   HumiditySensor sensor = advancedSensor;
    System.out.println(advancedSensor.purpose());
    System.out.println(sensor.purpose());
}
```

Metodų perrašymas

Vaikinėje klasėje galima pasiekti net ir perrašytus tėvinius metodus pasitelkiant raktažodį ***super***

Code Academy
Programuok savo ateitį!

Metodų perrašymas

```
public class AdvancedHumiditySensor extends HumiditySensor {
   @Override
    public String purpose() {
        return "precisely measures humidity";
    }
    public String parentPurpose() {
        return super.purpose();
    }
}
public static void main(String[] args) {
    AdvancedHumiditySensor advancedSensor = new AdvancedHumiditySensor();
    System.out.println(advancedSensor.purpose());
    System.out.println(advancedSensor.parentPurpose());
}
```

Code Academy
Programuok savo ateitį!

Metodų perrašymas

Norint išvengti paveldimumo, gali būti naudojamas raktažodis ***final***. Klasės su raktažodžiu ***final*** negali būti paveldėtos, o ***final*** metodai negali būti perrašyti

Code Academy
Programuok savo ateitį!

Abstrakčios klasės

Abstrakti klasė – tai klasė, turinti raktažodį ***abstract***.

- Abstrakčios klasės negali būti inicializuotos.
- Abstrakčios klasės turi abstrakčius metodus.

Paveldėjus abstrakčią klasę, vaikinė klasė privalo įgyvendinti visus abstrakčius tėvinės klasės metodus.

Code Academy
Programuok savo ateitį!

Abstrakčios klasės

Abstrakčios klasės turėtų būti taikomos naudojant tą patį kodą keliose, glaustai susijusiose klasėse

Code Academy
Programuok savo ateitį!

Abstrakčios klasės

Siuntų tarnybos sistema apdoroja įvairias siuntas. Tai gali būti paprasta *siunta, skubi siunta, didelio dydžio siunta*.
Visos šios siuntos turi bendrus parametrus – ***gavėjo adresą*** ir ***išsiuntimo datą***. Taip pat apie siuntas turėtų būti žinoma **siuntos gavimo data, siuntimo būdas**. Ši informacija būdinga kiekvienai siuntai, tačiau kiekvienam tipui ji bus skirtinga

Code Academy
Programuok savo ateitį!

Abstrakčios klasės

```
public abstract class Shipment {
    private final String address;
    private final LocalDateTime shippingDate;

    public Shipment(String address, LocalDateTime shippingDate) {
        this.address = address;
        this.shippingDate = shippingDate;
    }
    public abstract LocalDateTime expectedDelivery();
    public abstract String shipmentMethod();

    public String getAddress() {
        return address;
    }
    public LocalDateTime getShippingDate() {
        return shippingDate;
    }
}
```

Abstrakčios klasės

```
public class StandardShipment extends Shipment {
        public StandardShipment(String address, LocalDateTime shippingDate) {
            super(address, shippingDate);
        }
        @Override
        public LocalDateTime expectedDeliver() {
            return getShippingDate().plusDays(10);
        }
        @Override
        public String shipmentMethod() {
            return "by van";
        }
    }
```

Code Academy
Programuok savo ateitį!

Abstrakčios klasės

```
public class LargeShipment extends Shipment {
    public LargeShipment(String address, LocalDateTime shippingDate) {
        super(address, shippingDate);
    }
    @Override
    public LocalDateTime expectedDelivery() {
        return getShippingDate().plusDays(20);
    }
    @Override
    public String shipmentMethod() {
        return "by train";
    }
}
```

Code Academy
Programuok savo ateitį!

Abstrakčios klasės

```
public class UrgentShipment extends Shipment {
    public UrgentShipment(String address, LocalDateTime shippingDate) {
        super(address, shippingDate);
    }
    @Override
    public LocalDateTime expectedDelivery() {
        return getShippingDate().plusDays(3);
    }
    @Override
    public String shipmentMethod() {
        return "by plane";
    }
}
```

Code Academy
Programuok savo ateitį!

Sąsajos

Kompiuterių inžinerijoje dažni atvejai, kai atskiros programuotojų grupės turi sudaryti „kontraktus“, apibrėžiančius, kaip jų programinė įranga bendrauja tarpusavyje.
Kiekviena grupė turėtų gebėti kurti savo programinį kodą, nežinodami, kaip parašytas kitų grupių kodas.
Sąsajos (***interfaces***) Java programavimo kalboje ir yra tokie kontraktai

Code Academy
Programuok savo ateitį!

Sąsajos

Sąsajos gali turėti abstrakčius metodus bei konstantas (kintamuosius su public, static, final raktažodžiais). Taip pat nuo 8 Javos versijos, sąsajos gali turėti statinius metodus, bei ***default*** metodus – metodus su numatytu įgyvendinimu

Sąsajos

Sąsajos ypatingos tuo, kad klasės gali įgyvendinti neribotą kiekį sąsajų.
Abstrakčios klasės taip pat gali įgyvendinti sąsajas, tačiau sąsajų metodai gali likti abstraktūs.
Tokiu atveju sąsajų metodus turės įgyvendinti klasės, paveldinčios minėtą abstrakčią klasę

Code Academy
Programuok savo ateitį!

Abstrakti klasė ar sąsaja? Ką pasirinkti?

Naudokite abstrakčias klases kai:

- norite dalintis kodu tarp kelių, glaustai susijusių klasių
- jums reikia laukų, kurie nebūtų konstantos

Code Academy
Programuok savo ateitį!

Abstrakti klasė ar sąsaja? Ką pasirinkti?

Naudokite sąsajas kai:

- tikitės, kad klasės, įgyvendinančios jūsų sąsają, nebus glaustai susijusios
- norite užtikrinti kontraktą, bet jo įgyvendinimo detalės nėra svarbios
- norite pasinaudoti galimybe paveldėti kelias sąsajas

Taisyklingos praktikos

Programuojant reikėtų naudoti pačią žemiausią tėvinę klasę iš paveldėjimo grandinės, teikiančią norimą funkcionalumą.
Nėra reikalo naudoti vaikinį objekto tipą, jei visi reikalingi metodai būdingi tėviniam duomenų tipui.
Pavyzdžiui, nekurkite ***TemperatureSensor*** tipo objektų, jei jums tereikia ***Sensor*** tipo metodų

Code Academy
Programuok savo ateitį!

Taisyklingos praktikos

Tvarkingi metodai grąžina sąsajas. Klientui nereikia žinoti, ar jis gauna ***ArrayList*** ar ***LinkedList*** įgyvendinimą, jam svarbu kad gautas objektas tenkintų ***List*** sąsajos kontraktą