NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020
EIROPAS SAVIENĪBA
Eiropas Sociālais fonds
IEGULDĪJUMS TAVĀ NĀKOTNĒ

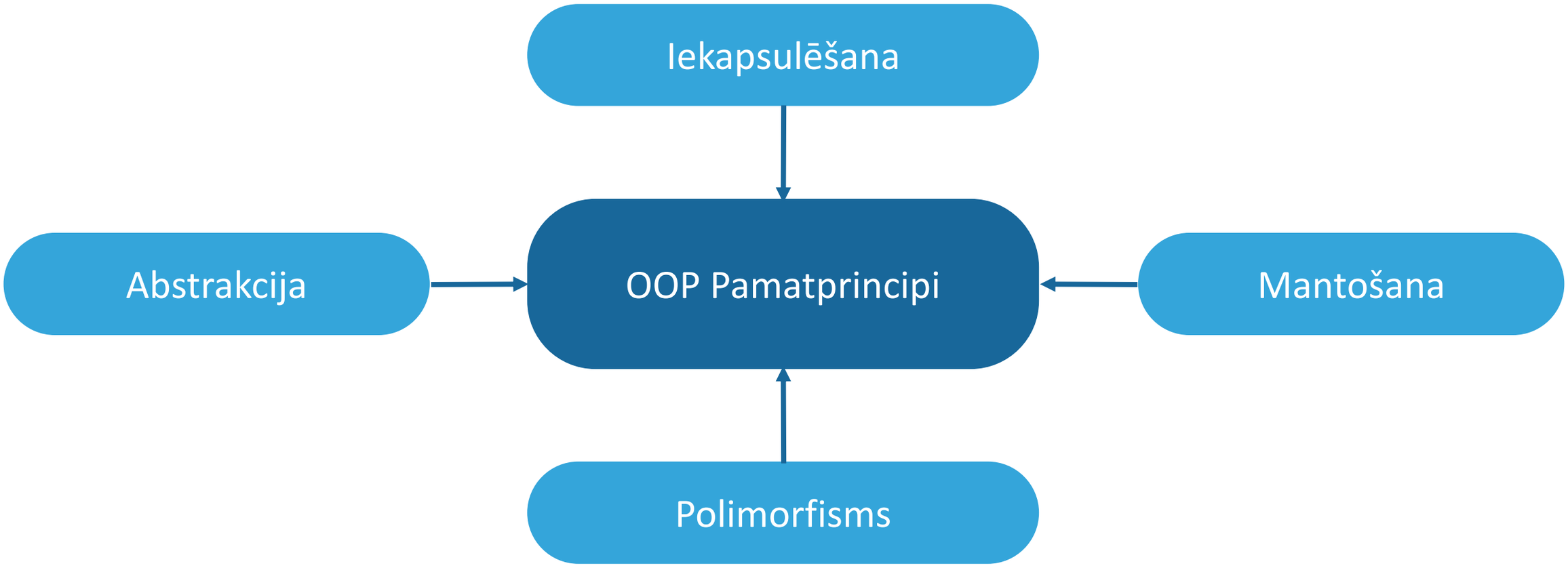
Java programmēšanas pamati

8. NODARBĪBA

KLAŠU HIERARHIJA

OBJEKT-ORIENTĒTĀS PROGRAMMĒŠANAS PRINCIPI

ČETRI OBJEKT-ORIENTĒTĀS PROGRAMMĒŠANAS (OOP) PĪLĀRI



MANTOŠANA

MANTOŠANA

- Procesu, kurā viena klase pārņem citas klases elementus (metodes un laukus) no jau eksistējošas klases, sauc par mantošanu
- Mērķis ir nodrošināt koda atkārtotu izmantošanu, lai klasei būtu jāraksta tikai unikālas funkcijas

JAVA MANTOŠANAS TIPI

TIEŠĀ MANTOŠANA (SINGLE INHERITANCE)

- Attiecas uz bērna un vecāka klašu attiecībām, kur klase paplašina citu klasi

NETIEŠĀ MANTOŠANA (MULTILEVEL INHERITANCE)

HIERARHISKĀ MANTOŠANA (HIERARCHIAL INHERITANCE)

- Attiecas uz bērna un vecāka klašu attiecībām, kur vairāk kā viena klase paplašina vienu un to pašu klasi

JAUKTĀ MANTOŠANA (HYBRID INHERITANCE)

- Vairāk kā viena mantošanas tipa kombinācija vienā programmā

KONCEPTS - MANTOŠANA

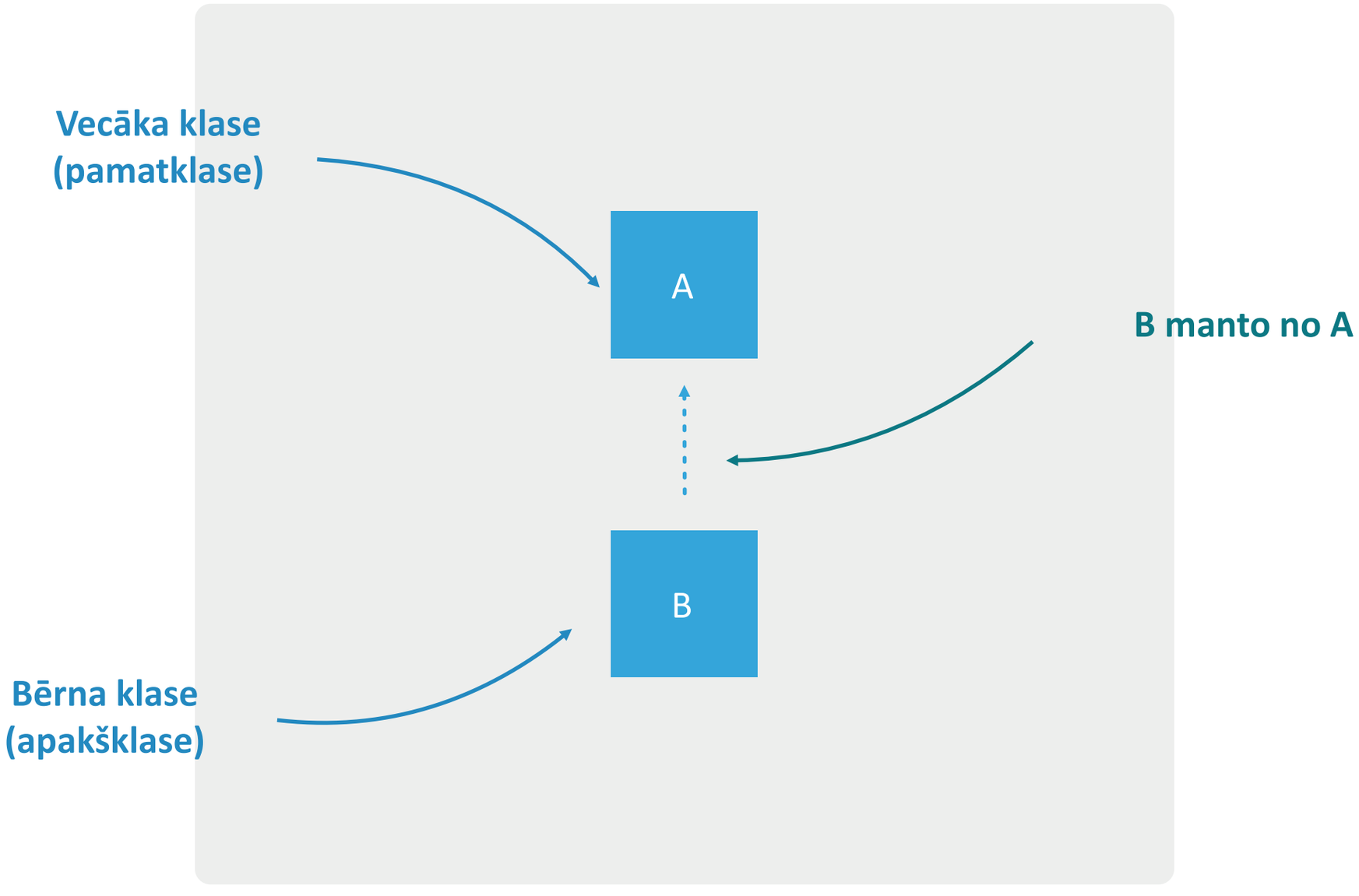
BĒRNA KLASE

- Klase, kas paplašina citas klases pazīmes, sauc par bērnu klasi, apakšklasi jeb atvasināto klasi

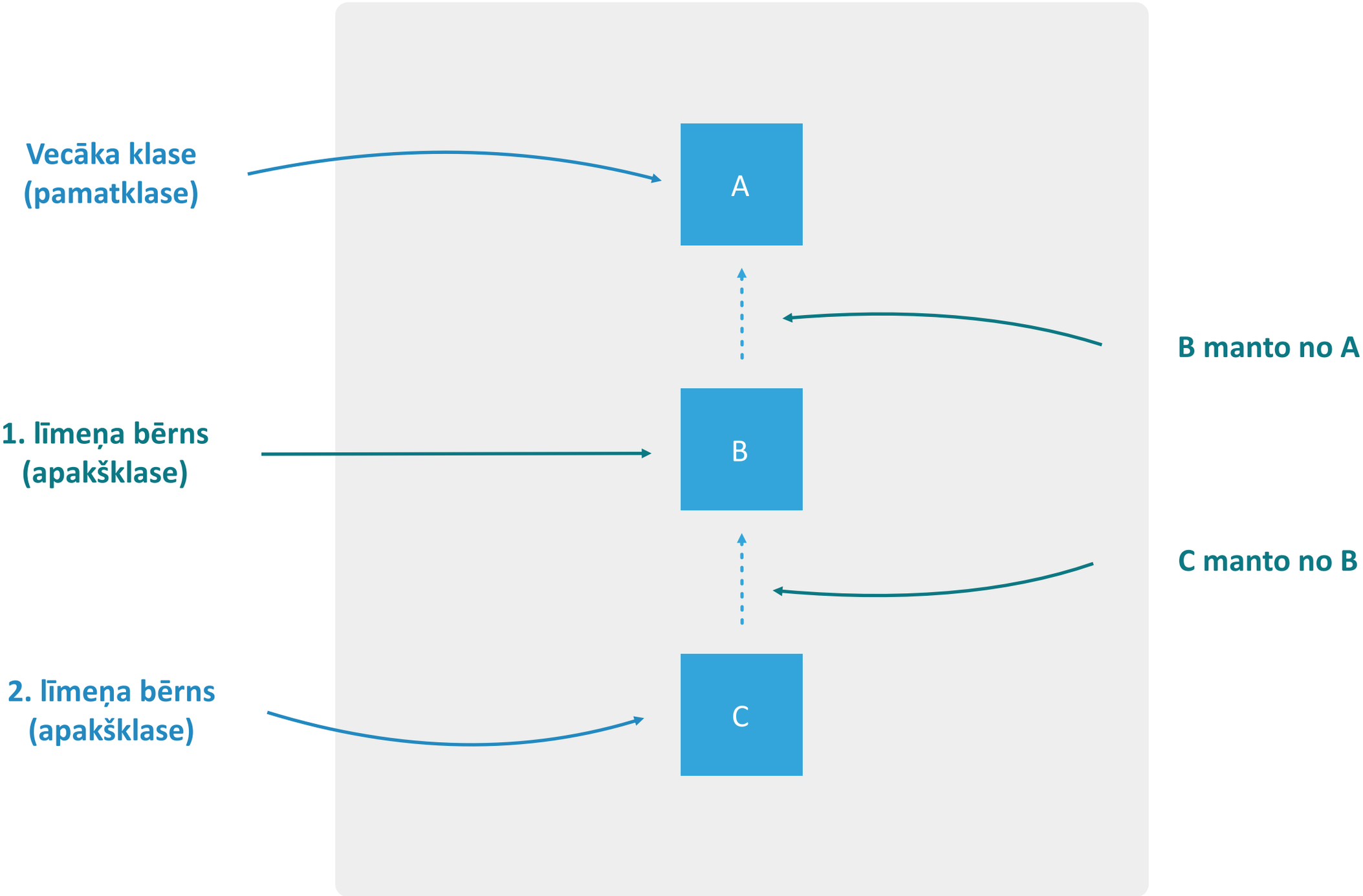
VECĀKA KLASE

- Klasi, kuras īpašības un funkcijas pārmanto cita klase, sauc par vecāka klasi, superklasi jeb bāzes klasi

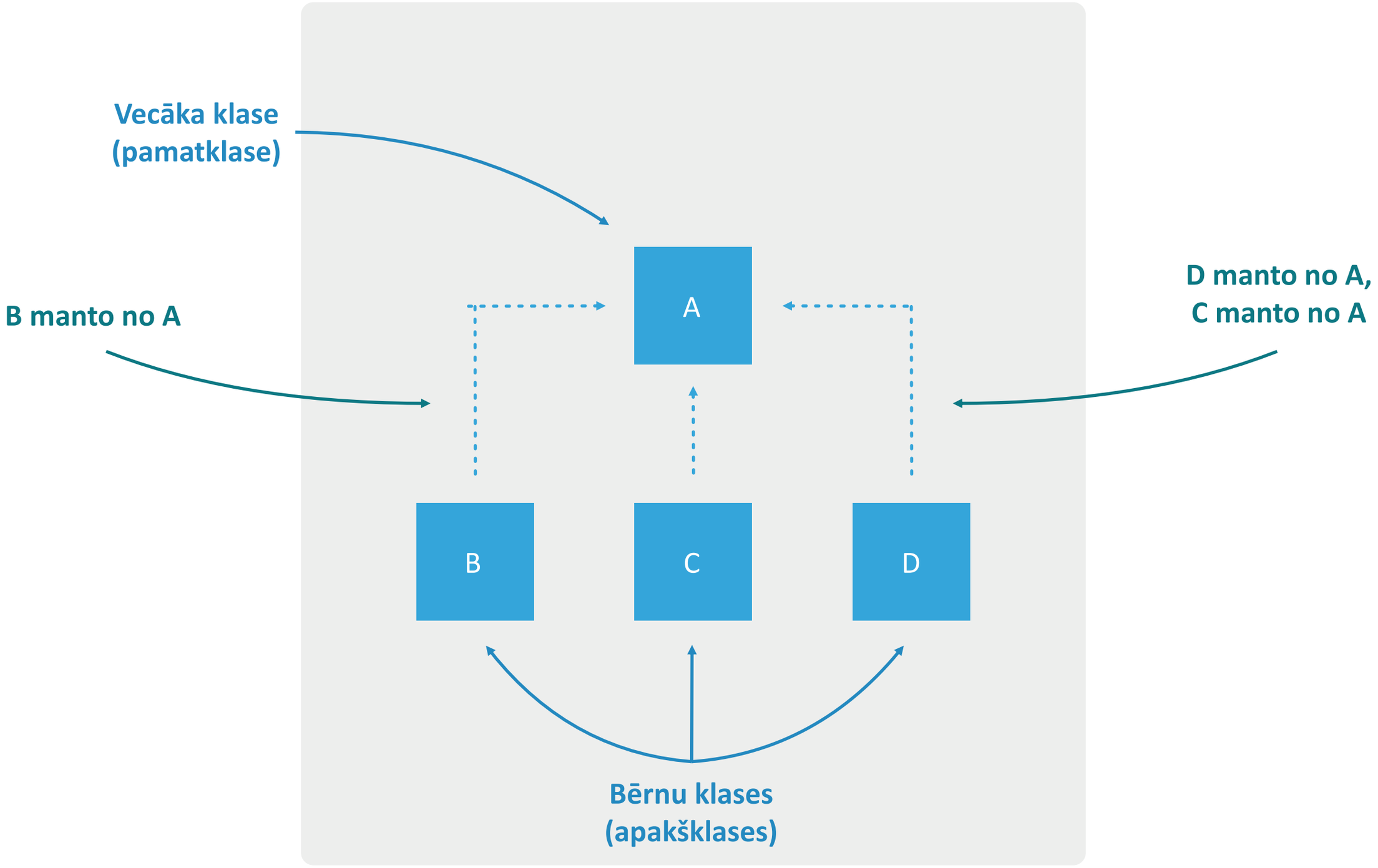
TIEŠA MANTOŠANA



NETIEŠA MANTOŠANA



HIERARHISKA MANTOŠANA



1. MANTOŠANA: PIEMĒRS

Operators “protected” ļauj apakšklasēm piekļūt elementam

```
public class Bicycle {

    protected String brand;
    protected int speed;

    public Bicycle(String brand, int speed) {
        this.brand = brand;
        this.speed = speed;
    }

    public void accelerate() {
        this.speed++;
    }

    public void decelerate() {
        this.speed--;
    }

    @Override
    public String toString() {
        return "Bicycle{" +
                "brand='" + brand + '\'' +
                ", speed=" + speed +
                '}';
    }

}
```

2. MANTOŠANA: PIEMĒRS

Apakšklase

Atslēgvārds norāda uz mantojamo klasi

Pamatklase

```
public class MountainBicycle extends Bicycle {

    protected int gear;

    public MountainBicycle(String brand, int speed, int gear) {
        super(brand, speed);
        this.gear = gear;
    }
```

Izsauc “vecāka” kostruktoru

```
    public void changeGear(int gear) {
        this.gear = gear;
    }

    @Override
    public String toString() {
        return "MountainBicycle{" +
                "gear=" + gear +
                ", brand='" + brand + '\'' +
                ", speed=" + speed +
                '}';
    }

}
```

3. MANTOŠANA: PIEMĒRS

Pirmkods

```
Bicycle bicycle = new Bicycle("Pinarello", 15);
MountainBicycle mountainBicycle = new MountainBicycle("BMC", 42, 2);

System.out.println(bicycle);
System.out.println(mountainBicycle);
```

Konsoles izvade

```
Bicycle{brand='Pinarello', speed=15}
MountainBicycle{gear=2, brand='BMC', speed=42}
```

4. MANTOŠANA: PIEMĒRS

Pirmkods

```
System.out.println("Pedal to the metal!");
mountainBicycle.accelerate();

System.out.println(bicycle);
System.out.println(mountainBicycle);
```

Konsoles izvade

```
Pedal to the metal!
Bicycle{brand='Pinarello', speed=15}
MountainBicycle{gear=2, brand='BMC', speed=43}
```

1. NOTEIKUMI UN IEROBEŽOJUMI

KATRAI KLASEI IR NOKLUSĒTĀ SUPERKLASE Object

- Ja klasei nav norādīta superklase, tad šīs klase netieši ir klases Object apakšklase
- Objekta klasei nav superklases

TIEŠĀS MANTOŠANAS PRINCIPI

- Superklasei var būt neierobežots skaits apakšklašu
- Apakšklasei var būt tikai viena superklase
- Java neatbalsta daudzklašu mantošanu (multiple inheritance), taču to ir iespējams daļēji panākt, izmantojot saskarnes (interfaces)

2. NOTEIKUMI UN IEROBEŽOJUMI

KONSTRUKTORI NETIEK MANTOTI

- Apakšklase manto no superklases visus locekļus (members): laukus, metodes un iekļautās klases (nested classes)
- Konstruktori nav locekļi, tāpēc apakšklases tos nemanto, bet no apakšklases var izsaukt virsklases konstruktoru

PRIVĀTO LOCEKĻU MANTOŠANA

- Apakšklase nemanto tās vecāku klases private locekļus
- Ja superklasei ir public vai protected metodes (piemēram, getters un seters), tās var izmantot arī apakšklase

MANTOŠANA: KOPSAVILKUMS

APAKŠKLASES VAR MANTOT LOCEKĻUS TĀDUS, KĀDI TIE IR, TOS MODIFICĒT, PASLĒPT VAI PAPILDINĀT AR JAUNIEM LOCEKĻIEM:

Definēt jaunus locekļus apakšklasē, kas neeksistē superklasē

- Lietot mantotos laukus tieši, tāpat kā pārējos klsases laukus
- Apakšklasē deklarēt jaunus laukus, kas neatrodas superklasē
- Apakšklasē uzrakstīt jaunu metodi, kurai ir tāds pats paraksts kā virsklasē, tādējādi to pārrakstot (overriding) (piemēram, equals(), toString())
- Apakšklasē deklarēt jaunas metodes, kas neatrodas superklasē
- Rakstīt apakšklases konstruktoru, kas netieši vai izmantojot atslēgvārdu super izsauc superklases konstruktoru

ABSTRAKCIJA

ABSTRAKCIJA

- Process, kurā jūs parādāt tikai būtiskos objekta datus un paslēpjat no lietotāja nesvarīgo objekta informāciju
- Ļauj abstrahēties no tiešā pielietojuma un drīzāk izklāstīt vispārīgo objektu funkcionalitāti
- Nosaka, ko objekts dara, nevis kā

ABSTRAKCIJA: KOPSAVILKUMS

ABSTRAKCIJA TIEK PANĀKTA IZMANTOJOT:

- Interfeisus (Interfaces)
  - Iespējams panākt pilnīgu abstrakciju
- Abstraktas klases (Abstract classes)
  - Iespējams panākt daļēju abstrakciju

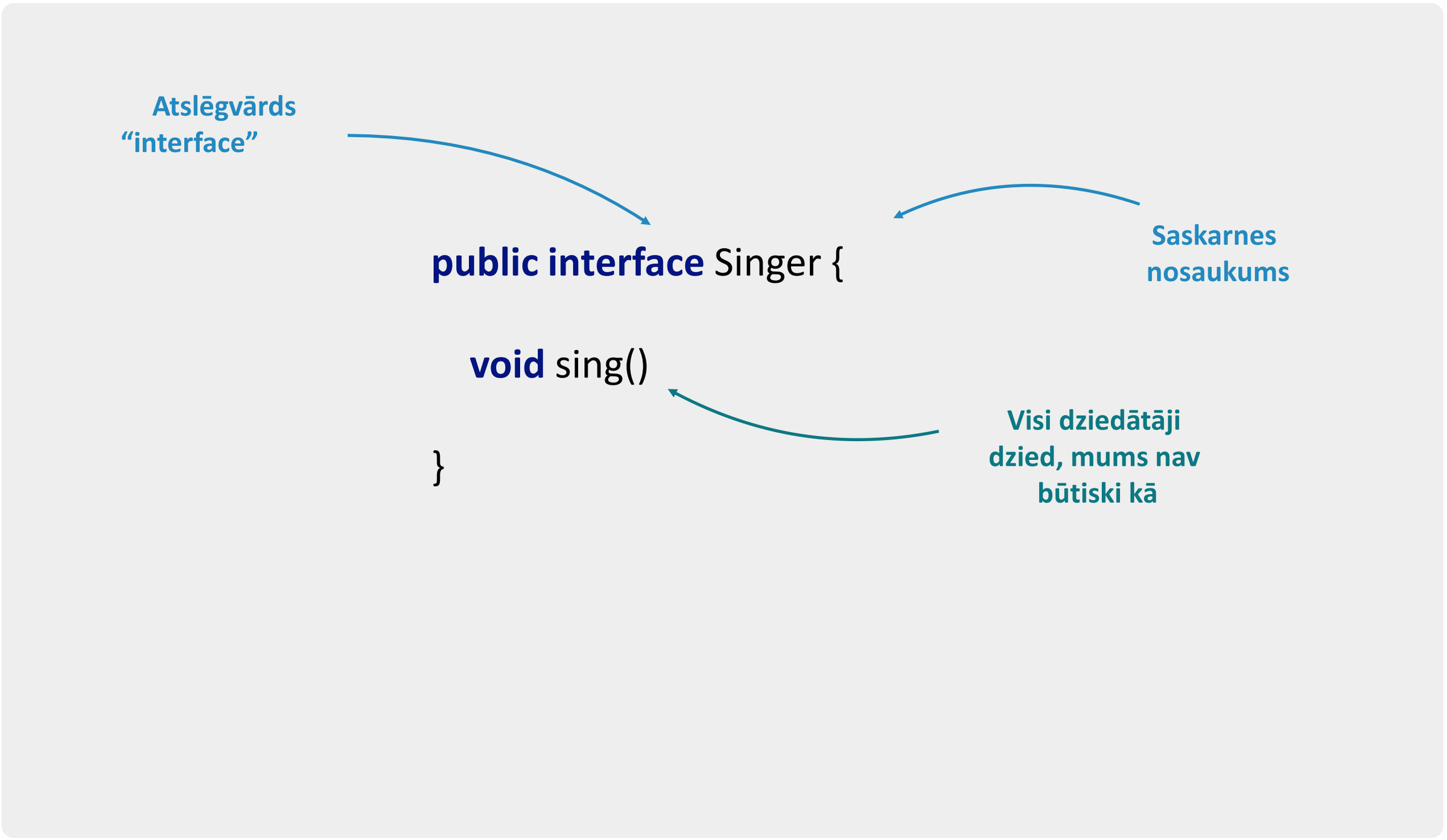
ABSTRAKTA KLASE

- Gandrīz kā klase, izņemot:
  - Var saturēt metožu parakstus bez pašas metodes implementācijas
  - Nevar tieši izveidot objektu

INTERFEISS

- Gandrīz kā klase, izņemot:
  - Iterfeiss var saturēt tikai metožu parakstus un laukus
- Interfeisa metodēs nav implementācijas, bet ir tikai paraksts (atgriežamais tips, nosaukums, parametri un izņēmumi)
- Apraksta objekta darbības
  - Interfeisa nosaukums nereti beidzās ar ‘-able’, postfikss (Comparable)

1. INTERFEISS: PIEMĒRS



2. INTERFEISS: PIEMĒRS

Atslēgvārds, kas nosaka, ka klase nodrošina norādītās saskarnes uzvedību

```
public class ElvisPresley implements Singer {

        .       .       .

        @Override
        public void sleep() {
                System.out.println("Love me tender, love me sweet");
        }

        .       .       .

}
```

Konkrēta saskarnes metodes implementācija

3. INTERFEISS: PIEMĒRS

```
public class BritneySpears implements Singer {

                    .       .       .

        @Override
        public void sleep() {
                System.out.println("Hit me baby one more time");
        }

                    .       .       .

}
```

4. INTERFEISS: PIEMĒRS

```
public class MichaelJackson implements Singer {

                                        .       .       .

        @Override
        public void sleep() {
                System.out.println("Billie Jean is not my lover");
        }

                                            .       .       .

}
```

ABSTRAKTA KLASE

- Gandrīz kā klase, izņemot:
  - Var saturēt metožu parakstus bez pašas metodes implementācijas
  - Nevar tieši izveidot objektu

1. ABSTRAKTAS KLASES PIEMĒRS

Atzīmē klasi ar atslēgvārdu "abstract"

```
public abstract class Shape {

    private String color;

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

    public String getColor() {
        return color;
    }

    abstract double area();

}
```

Apakšklasei ir Jābūt "vecāka" konstruktoram

Metodes "paraksts" (signature), kurš "bērna" klasei ir jāimplementē

2. ABSTRAKTAS KLASES PIEMĒRS

```
public class Circle extends Shape {

    private int radius;

    public Circle(String color, int radius) {
        super(color);
        this.radius = radius;
    }

    @Override
    double area() {
        return 3.14 * radius * radius;
    }

}
```

Paplašina klasi "Shape"

Apakšklases īpašība

Izsauc "vecāka" klases konstruktoru ar obligātajiem paremetriem

Katra konkrēta "forma" prot aprēķināt savu laukumu

3. ABSTRAKTAS KLASES PIEMĒRS

```
public class Rectangle extends Shape {

    private int width;
    private int height;

    public Rectangle(String color, int width, int height) {
        super(color);
        this.width = width;
        this.height = height;
    }

    @Override
    double area() {
        return width * height;
    }

}
```

Četrstūrim raksturīgās īpašības

1. INTERFEISS PRET ABSTRAKTA KLASE

METOŽU TIPI

- Interfeisam var būt tikai abstraktas metodes (no Java 8 atbalsta arī statiskās un noklusējuma metodes)
- Abstraktā klasē var būt gan abstraktas, gan neabstraktas metodes

FINAL MAINĪGIE

- Visi interfeisā definēties mainīgie ir final pēc noklusejuma
- Abstraktā klasē var būt ne tikai final mainīgie

2. INTERFEISS PRET ABSTRAKTA KLASE

MAINĪGĀ VEIDS

- Interfeisam ir tikai static un final mainīgie
- Abstraktā klasē varbūt jebkura kombinācija

IMPLEMENTĀCIJA

- Interfeisā nevar būt metodes implementācija
- Abstraktā klasē var būt metodes gan ar, gan bez implementācijas

3. INTERFEISS PRET ABSTRAKTA KLASE

MANTOŠANA PRET ABSTRAKCIJU

- Interfeisu var implementēt izmantojot vārdu “implements”
- Abstrakti klasi var papildināt izmantojot vārdu “extends”

DAUDZKĀRTĒJA IMPLEMENTĀCIJA

- Interfeiss var paplašināt (extend) tikai saskarni un tikai vienu Java saskarni
- Abstrakta klase var paplašināt (extend) vienu Java klasi un implementēt vairākas saskarnes

4. INTERFEISS PRET ABSTRAKTA KLASE

DATU DALĪBNIEKU PIEEJAMĪBA

- Interfeisa locekļiem piekļuves modifikators pēc noklusējuma ir public un to nevar mainīt
- Abstraktās klases locekļiem var būt jebkurš piekļuves modifikators (izņemot privātās abstraktās metodes)

4. INTERFEISS PRET ABSTRAKTA KLASE

DATU DALĪBNIEKU PIEEJAMĪBA

- Interfeisa locekļiem piekļuves modifikators pēc noklusējuma ir public un to nevar mainīt
- Abstraktās klases locekļiem var būt jebkurš piekļuves modifikators (izņemot privātās abstraktās metodes)

POLIMORFISMS

POLIMORFISMS

- Polimorfisms ir objekta spēja iegūt dažādas formas
- Metodes spēja darīt dažādas lietas, pamatojoties uz objektu, uz kuru tā darbojas
- Programmas darbības laikā tiek izvēlēta nepieciešamā implementācija, atkarībā no situācijas

1. POLIMORFISMS: PIEMĒRS

Pirmkods

```
Singer elvis = new ElvisPresley();
Singer jackson = new MichaelJackson();
Singer spears = new BritneySpears();

elvis.sing(); jackson.sing(); spears.sing();
```

Konsoles izvade

```
Love me tender, love me sweet
Billie Jean is not my lover
Hit me baby one more time
```

2. POLIMORFISMS: PIEMĒRS

Pirmkods

```
Singer[] singers = new Singer[2];
singers[0] = new ElvisPresley();
singers[1] = new BritneySpears();

for (Singer singer : singers) {
        singer.sing();
}
```

Konsoles izvade

```
Love me tender, love me sweet
Billie Jean is not my lover
```

3. POLIMORFISMS: PIEMĒRS

Pirmkods

```
Shape circle = new Circle("Red", 3);
Shape rectangle = new Rectangle("Blue", 2, 4);

System.out.println("Circle area = " + circle.area());
System.out.println("Rectangle area = " + rectangle.area());
```

Konsoles izvade

```
Love me tender, love me sweet
Billie Jean is not my lover
Hit me baby one more time
```

ATSAUCES

- https://stackify.com/oops-concepts-in-java/
- https://www.tutorialspoint.com/java/java_inheritance.htm
- https://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/
- https://www.geeksforgeeks.org/abstraction-in-java-2/
- http://tutorials.jenkov.com/java/interfaces.html
- https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

LATVIJAS UNIVERSITATE
ANNO 1919

VADĪBAS UN UZŅĒMĒJDARBĪBAS MĀCĪBU CENTRS