JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Ohjelmointi 2 – Luento 4

Rajapinta, tyyppiparametrit

Luennon sisältö

- Rajapinta
- Tyyppiparametrit

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Rajapinta

Rajapinta

- Rajapinta määrittää metodin tai metodien esittelyrivejä, joita luokan on tarjottava, mikäli se lupaa *toteuttaa* rajapinnan
- Rajapinta toimii sitovana sopimuksena
- Rajapinta ei määritä metodien toteutuksia (poikkeuksena ns. *default* metodit; oletuskäyttäytyminen ilman tilaa)
- Luokka toteuttaa rajapinnan avainsanalla `implements`

```
public interface Rajapinta {
}
```

```java
public interface Rajapinta {
    void metodi();
}
```

```java
public interface Rajapinta {
    void metodi1();
    void metodi2();
    void metodi3();
}
```

Rajapinta

- Rajapinta määrittää metodin tai metodien esittelyrivejä, joita luokan on tarjottava, mikäli se lupaa *toteuttaa* rajapinnan
- Rajapinta toimii sitovana sopimuksena
- Rajapinta ei määritä metodien toteutuksia (poikkeuksena ns. *default* metodit)
- Luokka toteuttaa rajapinnan avainsanalla `implements`

```java
public interface Rajapinta {
    void metodi1();
    void metodi2();
    void metodi3();
}
```

```java
public class Luokka {
}
```

```java
public interface Rajapinta {
    void metodi1();
    void metodi2();
    void metodi3();
}
```

```java
public class Luokka implements Rajapinta{
}
```

Class 'Luokka' must either be declared abstract or implement abstract method 'metodi1()' in 'Rajapinta' :1

```java
public interface Rajapinta {
    void metodi1();
    void metodi2();
    void metodi3();
}
```

```java
public class Luokka implements Rajapinta{
    @Override
    public void metodi1() {

    }

    @Override
    public void metodi2() {

    }

    @Override
    public void metodi3() {

    }
}
```

Rajapinta

- Rajapinta määrittää metodin tai metodien esittelyrivejä, joita luokan on tarjottava, mikäli se lupaa *toteuttaa* rajapinnan
- Rajapinta toimii sitovana sopimuksena
- Rajapinta ei määritä metodien toteutuksia (poikkeuksena ns. *default* metodit)
- Luokka toteuttaa rajapinnan avainsanalla `implements`
- Rajapinta ei lähtökohtaisesti ota kantaa siihen, miten metodit on teknisesti toteutettu

```java
public interface Rajapinta {
    void metodi1();
    String metodi2();
    boolean metodi3(int i);
}
```

```java
public interface Rajapinta {
    void metodi1();
    String metodi2();
    boolean metodi3(int i);
}
```

```java
public class Luokka implements Rajapinta{
    @Override
    public void metodi1() {

    }

    @Override
    public String metodi2() {

        return "";
    }

    @Override
    public boolean metodi3(int i) {
        return false;
    }
}
```

Rajapinta

- Luokka voi toteuttaa useita rajapintoja

```java
public interface Rajapinta {
    void metodi1();
    String metodi2();
    boolean metodi3(int i);
}
```

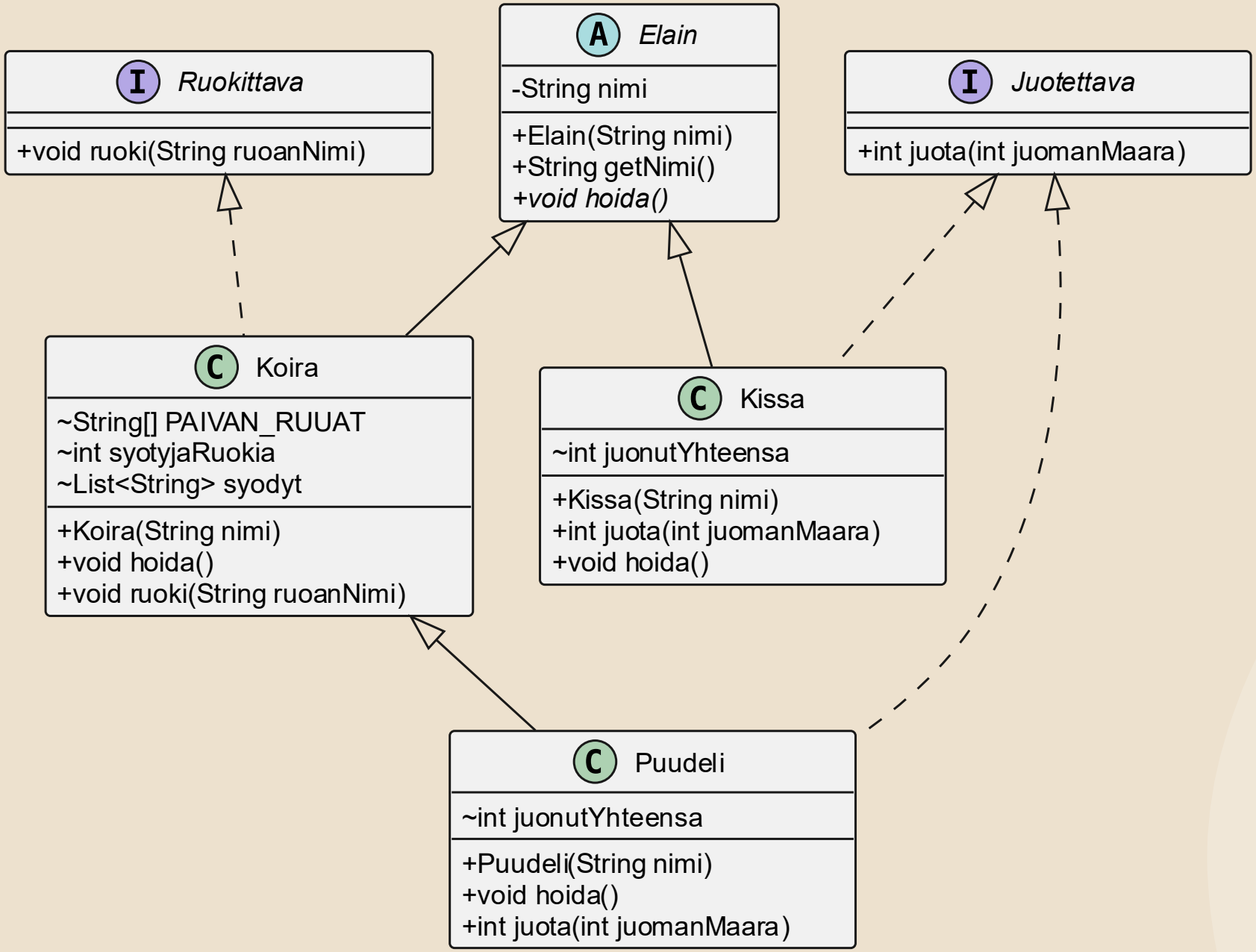
```java
public interface Rajapinta2 {
    void yllätys();
}
```

```java
public class Luokka implements Rajapinta, Rajapinta2 {
    @Override
    public void metodi1() {
    }

    @Override
    public String metodi2() {

        return "";
    }

    @Override
    public boolean metodi3(int i) {
        return false;
    }

    @Override
    public void yllätys() {
    }
}
```

Rajapinta

- ”is-capable”
- Rajapinta mahdollistaa yhtenevän kyvykkyyksien määrittelyn, vaikka luokat olisivat täysin erilaisia tai periytyisivät eri paikoista luokkahierarkiassa
- Kun ohjelmoija sitten käsittelee oliota rajapinnan kautta, hän voi luottaa siihen, että olio tarjoaa sovitun kyvykkyyden riippumatta siitä, mitä luokkaa olio edustaa
- Näin voidaan luottaa siihen että olio osaa jotakin, eikä käyttäjän tarvitse olla tietoinen miten olio hommansa hoitaa

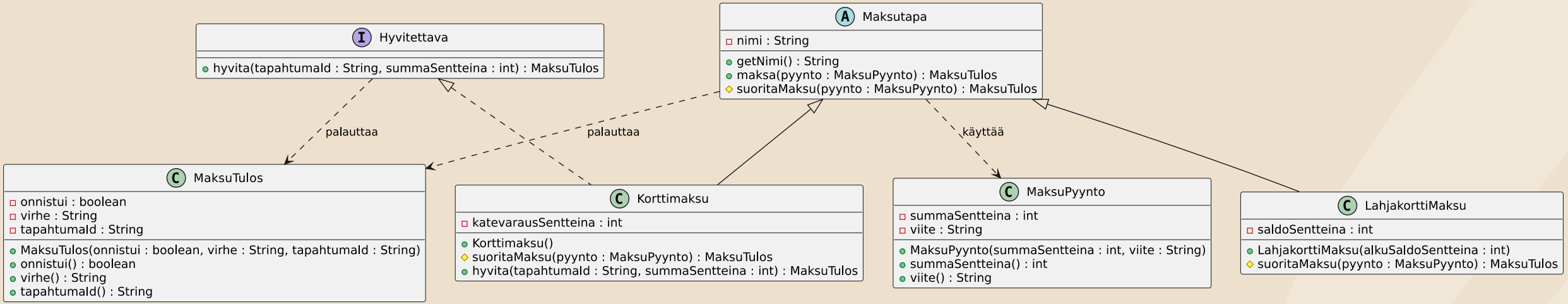
Demo



```java
public interface Rekisteroitava {
    String getRekisteriNumero();
    void asetaRekisteriNumero(String id);
}

public interface Ulkoilutettava {
    // Palauttaa true, jos lenkki meni hyvin
    boolean kayLenkilla(int minuutit);
}

public interface Harjattava {
    boolean onkoTakkuinen();
    void harjaa();
}
```

```java
public interface Rekisteroitava {
    String getRekisteriNumero();
    void asetaRekisteriNumero(String id);
}

public interface Ulkoilutettava {
    // Palauttaa true, jos lenkki meni hyvin
    boolean kayLenkilla(int minuutit);
}

public interface Harjattava {
    boolean onkoTakkuinen();
    void harjaa();
}
```

```java
public class Puudeli extends Koira implements Juotettava, Harjattava {
    // ... vanhat koodit ...

    @Override
    public void harjaa() {
        IO.println(getNimi() + ":n kiharat on nyt harjattu pöyheiksi!");
    }

    @Override
    public void hoida() {
        super.hoida(); // Tekee koiran hommat (syö)
        juota(1);      // Tekee juotavan hommat
        harjaa();      // Tekee harjattavan hommat
    }
}
```


Hyvitettava
hyvita(tapahtumaId : String, summaSentteina : int) : MaksuTulos
Maksutapa
nimi : String
getNimi() : String
maksa(pyynto : MaksuPyynto) : MaksuTulos
suoritaMaksu(pyynto : MaksuPyynto) : MaksuTulos
palauttaa
palauttaa
käyttää
MaksuTulos
onnistui : boolean
virhe : String
tapahtumaId : String
MaksuTulos(onnistui : boolean, virhe : String, tapahtumaId : String)
onnistui() : boolean
virhe() : String
tapahtumaId() : String
Korttimaksu
katevarausSentteina : int
Korttimaksu()
suoritaMaksu(pyynto : MaksuPyynto) : MaksuTulos
hyvita(tapahtumaId : String, summaSentteina : int) : MaksuTulos
MaksuPyynto
summaSentteina : int
viite : String
MaksuPyynto(summaSentteina : int, viite : String)
summaSentteina() : int
viite() : String
LahjakorttiMaksu
saldoSentteina : int
LahjakorttiMaksu(alkuSaldoSentteina : int)
suoritaMaksu(pyynto : MaksuPyynto) : MaksuTulos

Perintä vai rajapinta?

- Peukalosääntö: Mitä olio on? → perintä. Mitä olio osaa tehdä? → rajapinta
- Perintä kuvaa *is-a*-suhdetta. Aliluokka on yliluokan erikoistapaus
- Rajapinta kuvaa *capability*-suhdetta. Luokka toteuttaa jonkin toiminnallisen roolin
- Aliluokka perii tilan ja käyttäytymisen. Rajapinnassa ei ole tilaa (poikkeuksena default-metodit)
- Perintä toimii, kun yli- ja aliluokalla on samanlainen sisäinen tila. Luokkien välillä on luonnollinen hierarkia.

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Tyyppiparametri

Tyyppiparametri

- Parametrit mahdollistavat toiston vähentämisen yleistämällä ohjelman toimintaa erilaisille arvoille
- Funktion parametrit mahdollistavat saman funktion käyttämisen erilaisilla arvoilla (arvo argumenttina)
- Tyyppiparametrit mahdollistavat koodin yleistämisen erillisiin tietotyyppeihin (tyyppi argumenttina)
- Funktio, luokka tai rajapinta voi määritellä yhden tai useamman tyyppiparametrin
- Tyyppiparametri (tai -parametrit) määritellään kulmasulkeiden (`<>`) sisällä

```
public <T> void tulostaArvo(T arvo) {
  IO.println(arvo);
}
```

```java
public static <T> void tulostaArvo(T arvo) {
  IO.println(arvo);
}

void main() {
  tulostaArvo(1);
  tulostaArvo(3.14);
  tulostaArvo("Kissa");
  List<Integer> lista1 = List.of(1, 3, 5);
  tulostaArvo(lista1);
  List<Number> lista2 = List.of(1, 3.14, (byte)0);
  tulostaArvo(lista2);
}
```

Tyyppiparametri

- Tyyppiparametri voi olla mikä tahansa luokka tai rajapinta, jolle voidaan tarvittaessa asettaa rajoituksia
- Funktio tai luokka voi käyttää tyyppiparametreja määritellessään muuttujia, palautusarvoja tai muita tyyppiparametreja
- Funktio, luokka tai rajapinta, joka käyttää tyyppiparametreja, kutsutaan geneeriseksi

Esimerkkejä

- Esimerkki: Lisätään arvoja listaan (käytetään geneeristä kokoelmaa)
- Esimerkki: Tutkitaan, löytyykö tietty arvo listasta (luodaan geneerinen funktio)
- Esimerkki: Tutkitaan, kuinka monta kertaa tietty arvo esiintyy listassa (luodaan geneerinen funktio)

Useita tyyppiparametreja

- Tyyppiparametreja voi olla useita. Erotellaan pilkulla
- Esimerkki: Geneerinen luokka Pari<T1,T2>

Tyyppiparametrien rajoittaminen

- Tyyppiparametreille voidaan asettaa rajoitus tai rajoituksia
- Rajoitus voi olla luokka tai rajapinta
- Rajoitus määrittelee ylärajan tyypille, jota tyyppiparametri voi edustaa
- Käyttö `extends`-avainsanalla

```java
// Tyyppiparametri T voi olla vain Number-luokan alityyppi
<T extends Number> void tulostaLuku(T luku) {
    IO.println("Numero: " + luku.doubleValue());
}
```

```
// Tyyppiparametri T voi olla vain Number-luokan alityyppi
<T extends Number> void tulostaLuku(T luku) {
    IO.println("Numero: " + luku.doubleValue());
}

void main() {
    tulostaLuku(10);       // OK: Integer on Number
    tulostaLuku(3.14);     // OK: Double on Number
    // tulostaLuku("kissa"); // KÄÄNNÖSVIRHE: String ei ole Number
}
```

Tyyppiparametrien rajoittaminen

- Rajoituksia voidaan tarvittaessa asettaa myös useita käyttäen `&`-merkkiä

```
// Tyyppiparametri T voi olla vain luokka, joka on sekä Number että Comparable
<T extends Number & Comparable<T>> void vertaile(T a, T b) {
    if (a.compareTo(b) < 0) {
        IO.println(a + " on pienempi kuin " + b);
    } else if (a.compareTo(b) > 0) {
        IO.println(a + " on suurempi kuin " + b);
    } else {
        IO.println(a + " on yhtä suuri kuin " + b);
    }
}

void main() {
    vertaile(10, 20);      // OK: Integer on Number ja Comparable
    vertaile(3.14, 2.71);  // OK: Double on Number ja Comparable
    // vertaile("kissa", "koira"); // KÄÄNNÖSVIRHE: String ei ole Number
}
```