Informatika 2

Polymorfizmus, pretypovanie

doc. Ing. Ján Janech, PhD.
4. 3. 2024

ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Pojmy zavedené v 2. prednáške (1)

- polymorfizmus
- polymorfizmus a protokol

Pojmy zavedené v 2. prednáške (2)

- interface
- interface – Java
- interface – UML

Pojmy zavedené v 2. prednáške (3)

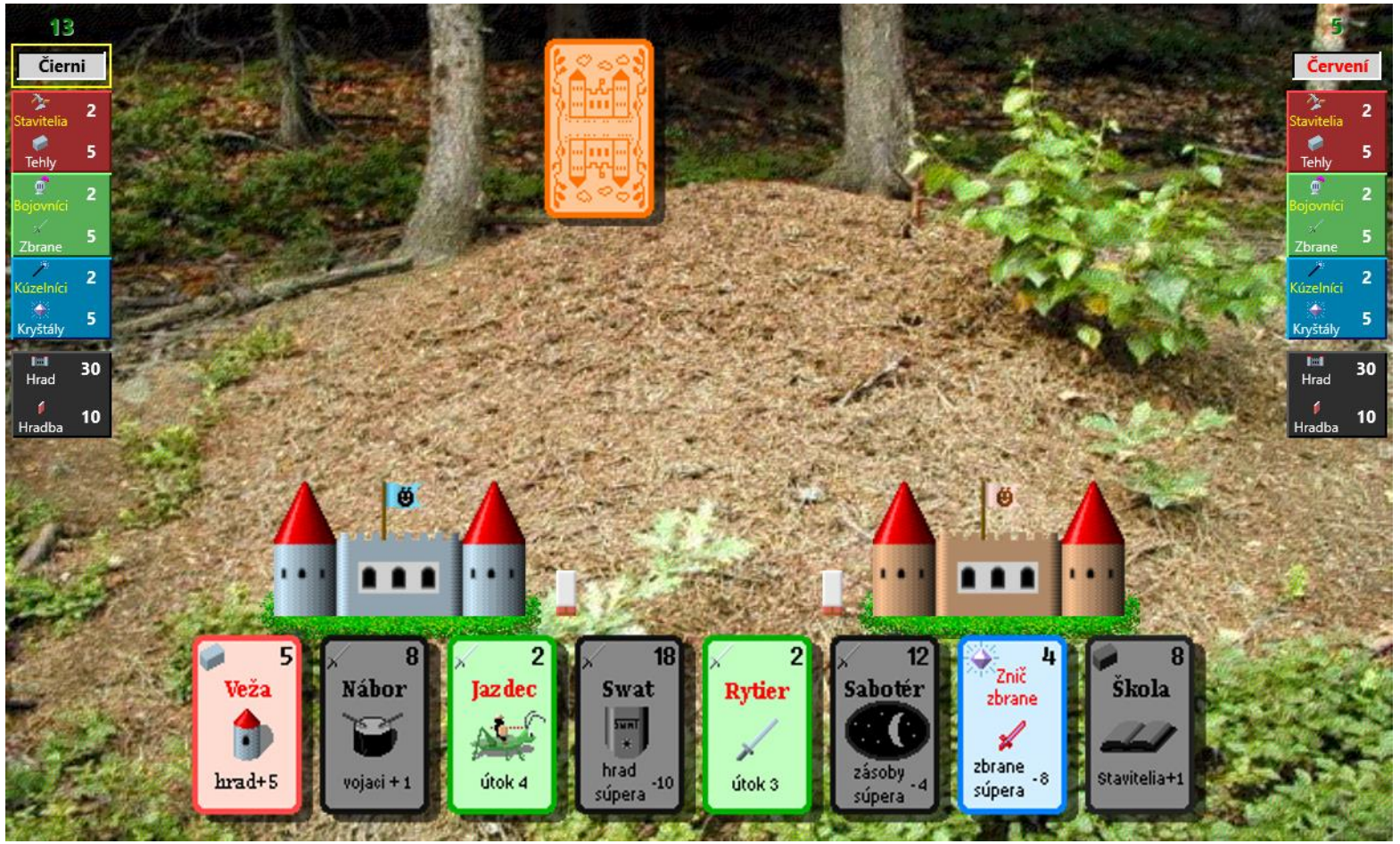
- typová kompatibilita – interface
- statický a dynamický typ

Cieľ prednášky

- pretypovanie – implicitné, explicitné
- implementácia viac interface v jednej triede
- bezpečné pretypovanie

- príklad: hra Mravenci

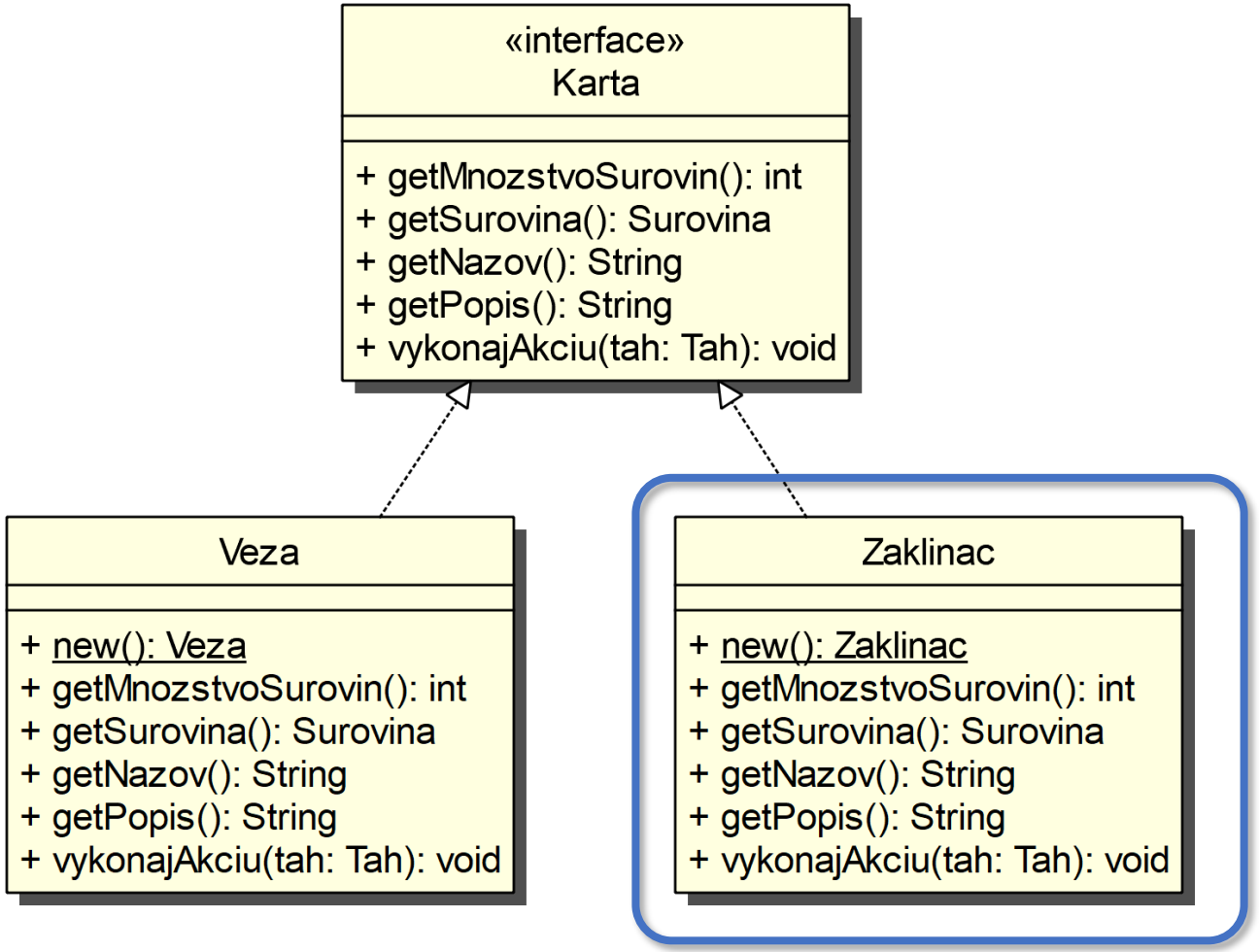
Hra Mravenci



Nová karta – Zaklínač

- nie je v pôvodnom balíčku
- akcia – zníži cenu všetkých kariet „Kliatba“ na ruke z 25 na 1 kryštál

Pridanie karty



Zmeny v karte Kliatba (1)

- zmenená metóda getMnozstvoSurovin() – vráti 1 ak je karta začarovaná
- pridaná metóda zacaruj() – označí kartu ako začarovanú

Zmeny v karte Kliatba (2)

```
private boolean zacarovana;
...
@Override
public int getMnozstvoSurovin() {
    if (this.zacarovana) {
        return 1;
    } else {
        return 25;
    }
}
...
public void zacaruj() {
    this.zacarovana = true;
}
```

Akcia v triede Zaklinac

- získa referencie na všetky kliatby na ruke
- každej pošle správu zacaruj()

Akcia v triede Zaklinac

```
@Override
public void vykonajAkciu(Tah tah) {
    Kliatba[] kliatby = tah.getRukaHraca().getKartyPodlaNazvu("Kliatba");

    for (var kliatba : kliatby) {
        kliatba.zacaruj();
    }
}
```

Akcia v triede Zaklinac – chyba pri preklade

```
@Override
public void vykonajAkciu(Tah tah) {
    Kliatba[] kliatby = tah.getRukaHraca().getKartyPodlaNazvu("Kliatba");

    for (var
        kliatba.zaciatok();
    }
}
```

Incompatible types: Karta[] cannot be converted to Kliatba[]

Akcia v triede Zaklinac – inak

```
@Override
public void vykonajAkciu(Tah tah) {
    Karta[] kliatby = tah.getRukaHraca().getKartyPodlaNazvu("Kliatba");

    for (var kliatba : kliatby) {
        kliatba.zacaruj();
    }
}
```

java: cannot find symbol
  symbol: method zacaruj()
  location: variable kliatba of type Karta

Akcia v triede Zaklinac – správne

```
@Override
public void vykonajAkciu(Tah tah) {
    Kliatba[] kliatby = tah.getRukaHraca().getKartyPodlaNazvu("Kliatba");

    for (var karta : kliatby) {
        var kliatba = (Kliatba)karta;
        kliatba.zacaruj();
    }
}
```

Pretypovanie

- zmena statického typu
- zmena množiny správ, ktoré je možné poslať

- implicitné
- explicitné

Implicitné pretypovanie

- automatické pretypovanie
- iba obmedzenie množiny správ
- vykonáva a kontroluje prekladač pri preklade
- statický typ musí byť typovo kompatibilný s cieľovým typom
- literál null sa dá implicitne pretypovať na ľubovoľný objektový typ
- minulý semester pod názvom Implicitná konverzia
  - implicitné pretypovanie je všeobecnejší názov

```
Karta karta = new Kliatba();
```

Explicitné pretypovanie

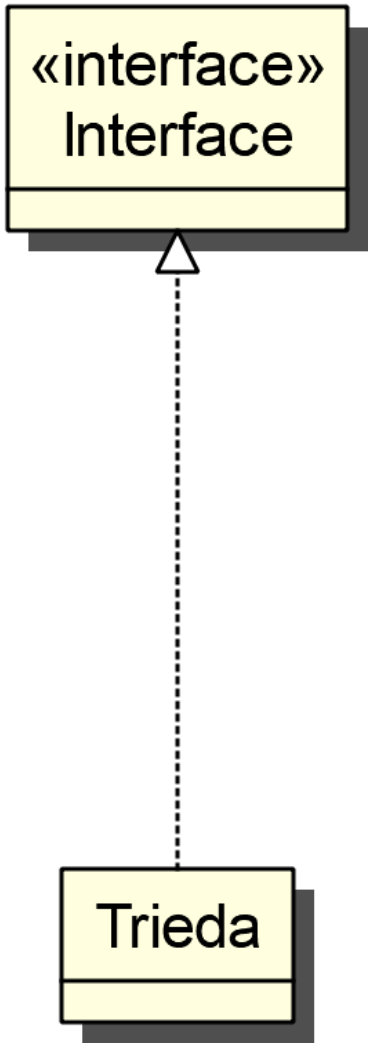
- pretypovanie „na požiadanie“
- rozšírenie, alebo výmena množiny správ
- prekladač robí iba čiastočnú kontrolu
- vykonáva a kontroluje JVM za behu programu
- dynamický typ musí byť typovo kompatibilný s cieľovým typom
- hodnota null sa dá explicitne pretypovať na ľubovoľný objektový typ
- minulý semester pod názvom Explicitná konverzia
  - explicitné pretypovanie je všeobecnejší názov

```
Kliatba kliatba = (Kliatba)karta;
```

Pretypovanie (1)

Implicitné
I

Explicitné
E

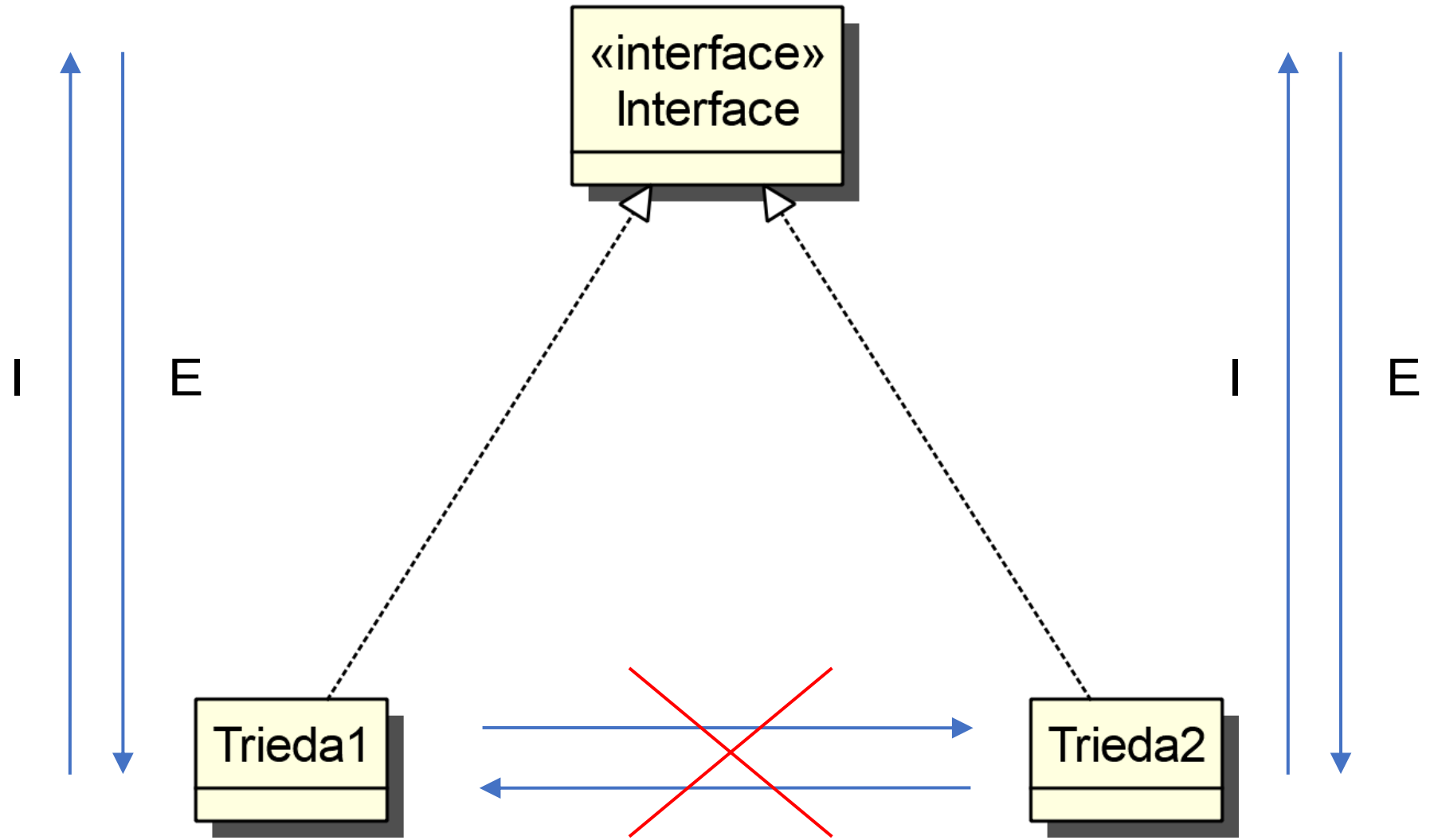


Pretypovanie (2)

```
Karta karta;
Kliatba kliatba = new Kliatba();


karta = kliatba; // OK
kliatba = karta; // Chyba pri preklade
kliatba = (Kliatba) karta; // OK
```

Pretypovanie (3)



Pretypovanie (4)

```
Vozidlo vozidlo;
Auto auto = new Auto();
Bicykel kolo = new Bicykel();

vozidlo = auto; // OK
auto = kolo; // Chyba pri preklade
auto = (Auto) kolo; // Chyba pri preklade
auto = (Auto) vozidlo; // OK
kolo = (Bicykel) vozidlo; // Chyba za behu
```

Nová karta Zázračná fazuľka

- nie je v pôvodnom balíčku
- akcia – žiadna, nemá zmysel vykladať
- efekt – kým je na ruke, v každom ťahu sa hradba zvýši o 1

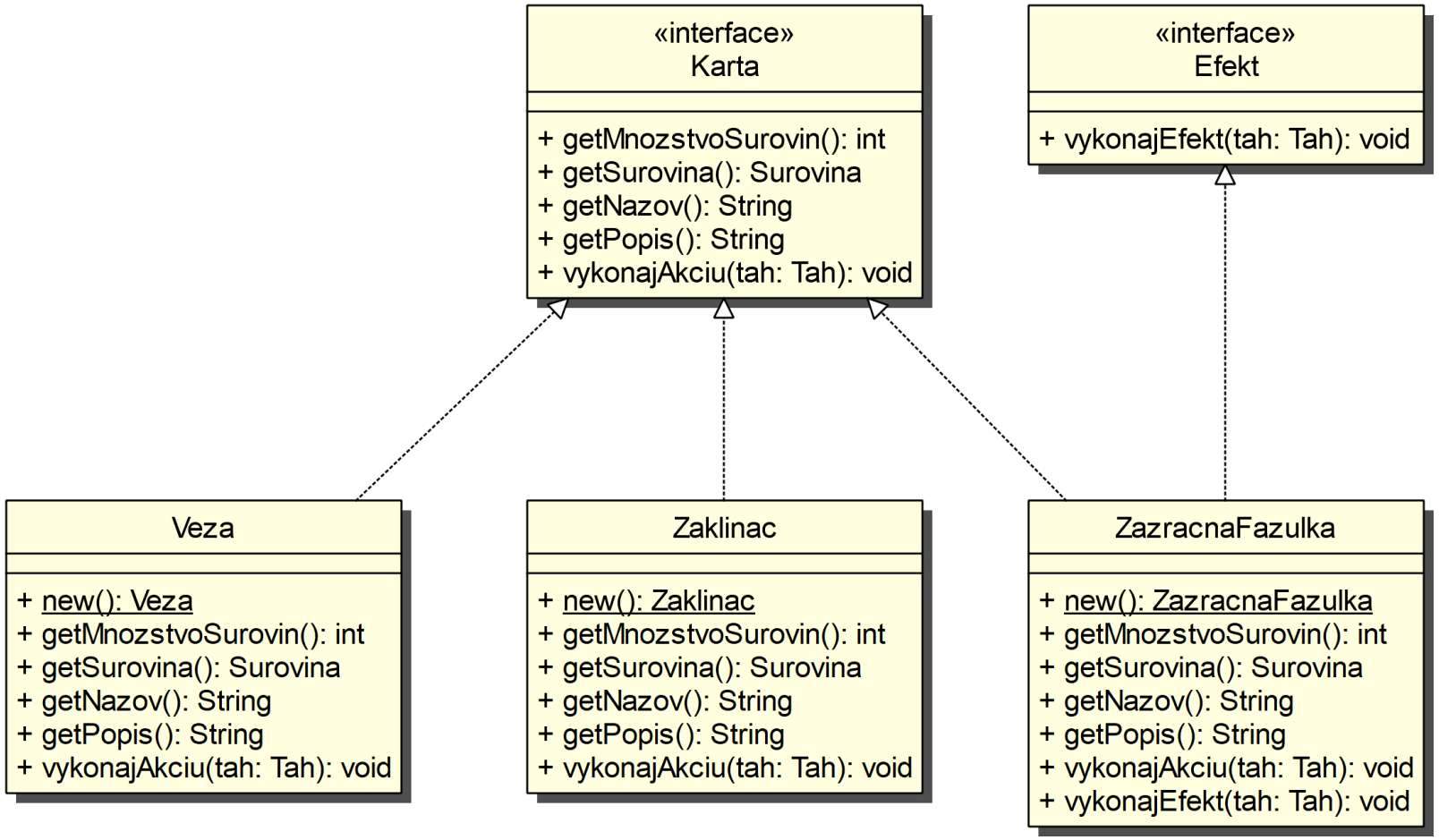
Nový princíp efekty

- efektová karta môže (nemusí mať) akciu
- efekt sa aplikuje na začiatku ťahu pre všetky efektové karty
- aplikovanie efektu nestojí žiadne suroviny

Riešenie

- Zázračná fazuľka je:
  - karta (dá sa držať na ruke a získať z balíčka)
  - efekt (dá sa aplikovať v každom ťahu)

UML riešenia



Implementácia viac interface

- trieda môže implementovať ľubovoľný počet interface (0...n)
- kľúčové slovo implements len jeden krát
- oddeľovač – čiarka

```
public class ZazracnaFazulka implements Karta, Efekt
```

Trieda ZazracnaFazulka (1) – hlavička

```
public class ZazracnaFazulka implements Karta, Efekt {

    ...
}
```

Trieda ZazracnaFazulka (2) – implementácia Karta

```
@Override
public int getMnozstvoSurovin() {
    return 0;
}

@Override
public Surovina getSurovina() {
    return Surovina.KRYSTAL;
}

@Override
public void vykonajAkciu(Tah tah) {

}
```

Trieda ZazracnaFazulka (3) – implementácia Karta

```
@Override
public String getNazov() {
    return "Čarovná fazuľka";
}

@Override
public String getPopis() {
    return "EFEKT: Hradba +1/ťah";
}
```

Trieda ZazracnaFazulka (4) – implementácia Efekt

```
@Override
public void vykonajEfekt(Tah tah) {
    tah.getHradHraca().zmenVyskuHradieb(+1);
}
```

Začlenenie efektu do hry

- len implementovať interface nestačí
- treba posielať správy
  - každej karte pošleme na začiatku ťahu správu vykonajEfekt

Vykonanie efektov v triede Ruka

```
public void vykonajEfekty(Tah aktualnyTah) {
    for (var karta : this.karty) {
        var efekt = (Efekt)karta;
        efekt.vykonajEfekt(aktualnyTah);
    }
}
```

Behová chyba pri spustení (1)

```
Exception ClassCastException: class Hradba cannot be cast to class Efekt
        at Ruka.vykonajEfekty(Ruka.java:34)
        at Hra.zaciatokTahu(Hra.java:84)
        at Hra.<init>(Hra.java:25)
        at Main.main(Main.java:9)
```

Behová chyba pri spustení (2)

- v hlásení: Inštancia triedy Hradba sa nedá explicitne pretypovať na Efekt
  - čítajte: trieda Hradba neimplementuje interface Efekt

- karta hradba nemá efekt, nemá ani implementovať interface

Problém

```
public void vykonajEfekty(Tah aktualnyTah) {
    for (var karta : this.karty) {
        var efekt = (Efekt)karta;
        efekt.vykonajEfekt(aktualnyTah);
    }
}
```

Riešenie

- Pretypovať na Efekt len tie karty, ktoré implementujú interface
  - vetvenie
  - operátor instanceof

Správna implementácia vykonajEfekty

```
public void vykonajEfekty(Tah aktualnyTah) {
    for (var karta : this.karty) {
        if (karta instanceof Efekt) {
            var efekt = (Efekt)karta;
            efekt.vykonajEfekt(aktualnyTah);
        }
    }
}
```

Operátor instanceof

```
prvyOperand instanceof druhyOperand
```

- prvyOperand – hodnota objektového typu
- druhyOperand – typ (trieda, enum, interface)

- vracia true ak
  - je prvyOperand inštanciou danej triedy
  - je prvyOperand inštanciou daného enumu
  - je prvyOperand inštanciou triedy implementujúcej daný interface
- kontroluje typovú kompatibilitu dynamického typu so zadaným typom

Bezpečné pretypovanie (1)

- operácia pretypovania funguje iba ak
  - je hodnota inštanciou danej triedy
  - je hodnota inštanciou daného enumu
  - je hodnota inštanciou triedy implementujúcej daný interface
- v opačnom prípade zlyhá a vyhodí behovú chybu
- => nutnosť testovania pomocou operátora instanceof

Bezpečné pretypovanie (2)

```
if (premenna instanceof Typ) {
    Typ pretypovana = (Typ)premenna;
    // prikazy po bezpecnom pretypovani
}
```

- príkazy sa vykonajú len ak sa dá premenná pretypovať na daný typ

Bezpečné pretypovanie – nový zápis od JDK 16

```
if (premenna instanceof Typ premenna) {
    // prikazy po bezpecnom pretypovani
}
```

- definovaná premenná v príkaze instanceof
- platnosť premennej iba v tele if-u

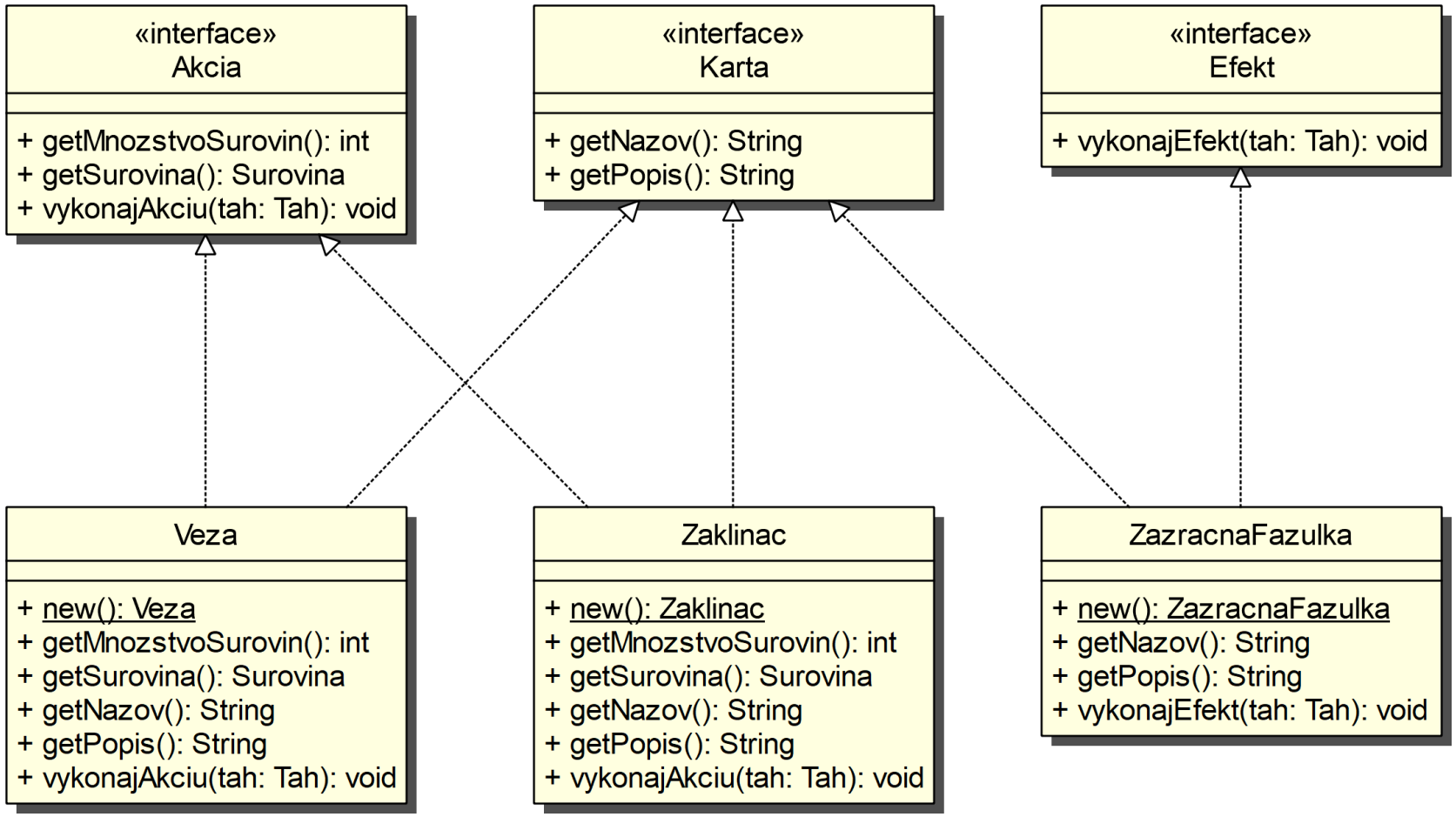
- (pattern matching)

Ďalšie zjednodušenie

- nie každá karta má akciu (zatiaľ je bez akcie ZazracnaFazulka)
- zbytočné implementovať akciu (a cenu v surovinách)

- zavedenie nového interface Akcia

Zavedenie interface Akcia – UML



Rozširovanie interface

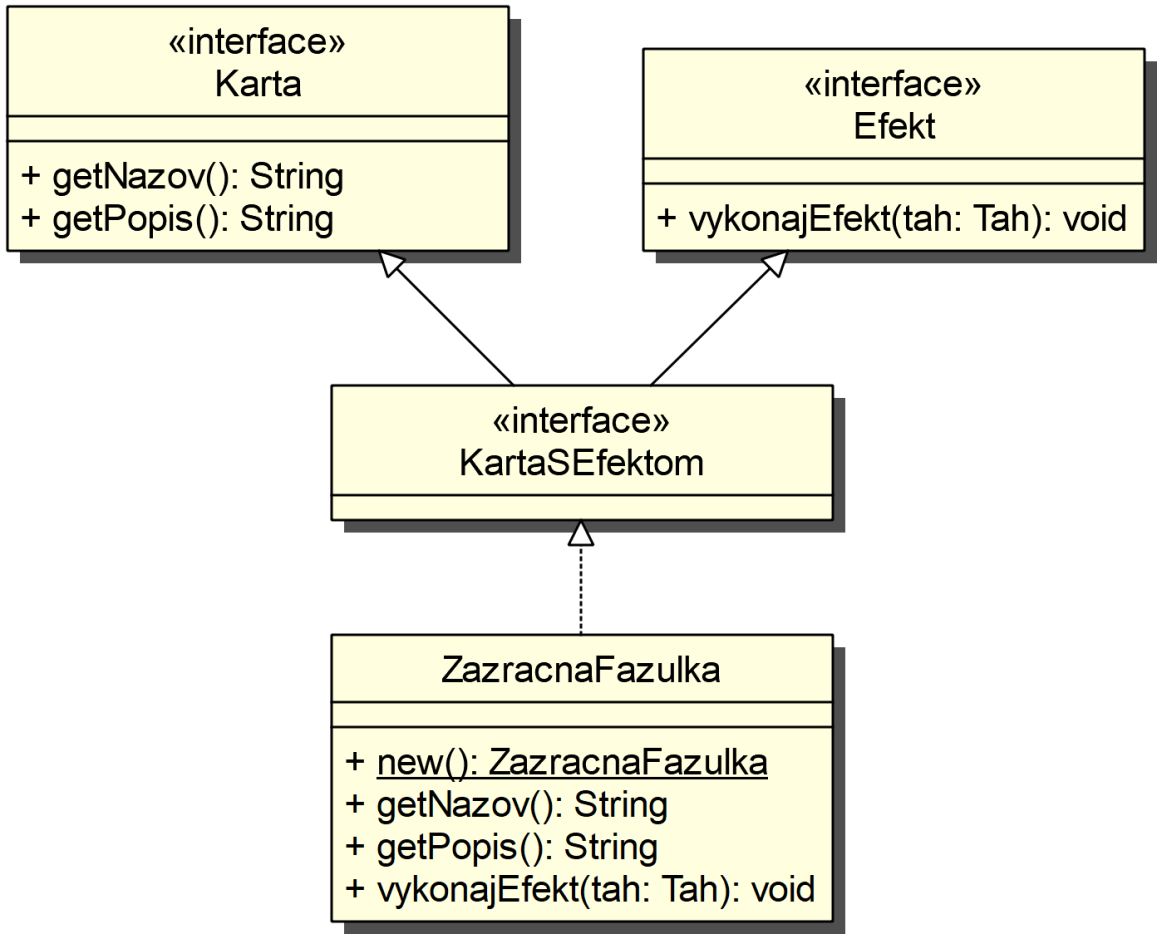
- možnosť označiť interface, ako rozšírenie jedného alebo viac interface
- kľúčové slovo extends v hlavičke interface
- Trieda, ktorá implementuje rozšírený interface, automaticky implementuje aj rozširované interface

- Napr.:

```
public interface KartaSEfektorm extends Karta, Efekt {

}
```

Príklad s kartou



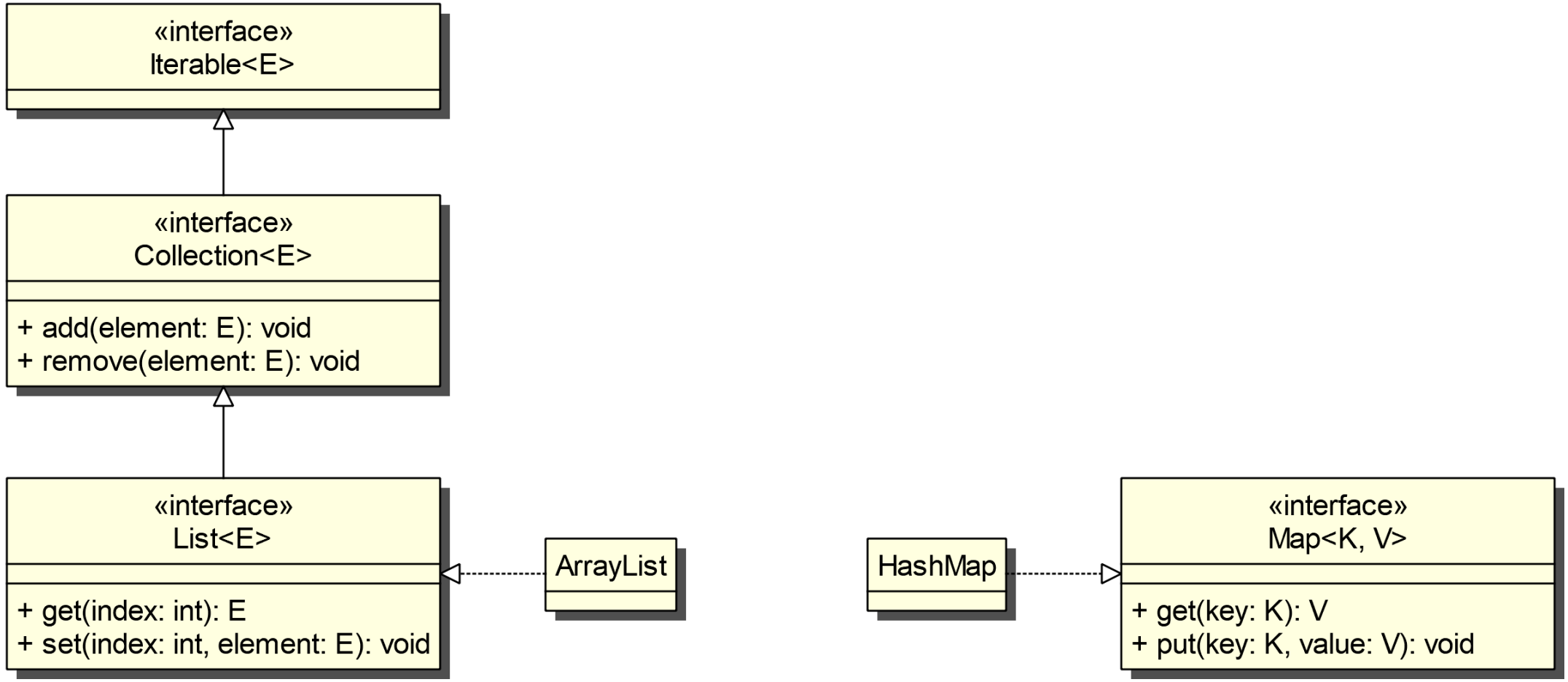
Interface v štandardnej knižnici

- v štandardnej knižnici veľké množstvo interface
- Implementované aj v triedach, ktoré poznáme

- napr. ArrayList implementuje (okrem iného):
  - List – umožňuje prístup cez index(get/set)
  - Collection – umožnuje pridávať/mazať prvky (add/remove)
  - Iterable – umožňuje prístup cez foreach
- HashMap implementuje (okrem iného):
  - Map – prístup k prvkom pomocou kľúča

- pole – nič
  - možnosť „obaliť“ do objektu implementujúceho List pomocou správy Arrays.asList(pole)

Kontajnery a interface – UML



Zlepšenie zapuzdrenia pomocou interface (1)

- minulý semester sme si spomínali problém
  - návratová hodnota metódy nemá vracať ArrayList, ak je komponentom
  - niekedy sa to ale zíde – kvôli prechádzaniu zoznamu pomocou foreach
- napr.:

```
public class SkupinaStudentov {
    private ArrayList<Student> studenti;
...
    public ArrayList<Student> getStudenti() {
        return this.studenti;
    }
}
```

Zlepšenie zapuzdrenia pomocou interface (2)

- riešenie umožňuje meniť ArrayList (komponent) cez návratovú hodnotu
- pravdepodobne sme chceli len umožniť prechádzanie cez for-each

- problém:
  - porušenie zapúzdrenia – dá sa zmierniť dokumentáciou
  - neprehľadné riešenie – bez štúdia dokumentácie sa nedá zistiť, že možnosť modifikácie bola zámerná

- pozn: niekedy skutočne môžete chcieť vrátiť kontajner, ale musí to byť vedomé a premyslené rozhodnutie

Zlepšenie zapuzdrenia pomocou interface (3)

- možnosť riešenia pomocou interface Iterable
- Napr.:

```
public class SkupinaStudentov {
    private ArrayList<Student> studenti;
...
    public Iterable<Student> getStudenti() {
        return this.studenti;
    }
}
```

Collections.unmodifiable*

- ak je v takomto prípade vyžadovaný interface List, Collection, Map, treba vrátiť nemodifikovateľný kontajner
  - napr. pri prístupe cez index

- správy triede Collections
  - unmodifiableMap – vytvorí z HashMap-u nemodifikovateľnú implementáciu Map
  - unmodifiableList – vytvorí z ArrayList-u nemodifikovateľnú implementáciu List
  - unmodifiableCollection – vytvorí z ArrayList-u nemodifikovateľnú implementáciu Collection

- pokus o zmenu – behová chyba
  - uviesť do dokumentácie !!!

Využitie Collections.unmodifiable*

- riešenie zoznamu študentov
  - môžeme vrátiť List
  - obalíme zoznam pomocou Collections.unmodifiableList

```
public class SkupinaStudentov {
    private ArrayList<Student> studenti;
...
    public List<Student> getStudenti() {
        return Collections.unmodifiableList(
            this.student
        );
    }
}
```