Programozási technológia

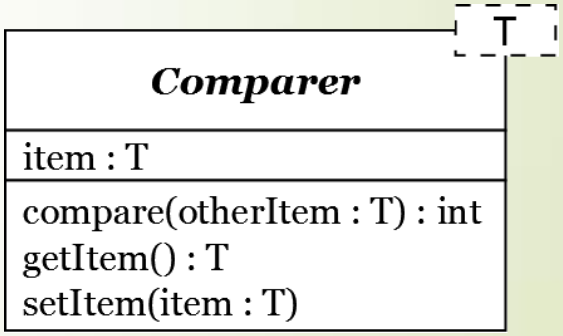
Generikus osztályok
Gyűjtemények

Dr. Szendrei Rudolf
ELTE Informatikai Kar
2018.

Generikus osztályok

- Javaban az UML paraméteres osztályainak a generikus (sablon) osztályok felelnek meg, ahol a paraméterekből generikus paramétereket készítünk.
- A generikus paraméterek csak osztálynevek lehetnek, amelyekkel a generikus osztály definíciójában paraméterezhető típusok adhatók meg.

```
public class Comparer<T>{
  private T   item;
  public  T   getItem(){}
  public void setItem(T item){...}
  public int  compare(T otherItem){...}
}
```



Generikus osztályok

- A generikus osztályok használatakor meg kell adni a generikus paraméterek konkrét értékeit (osztályneveket). Ekkor a generikus paraméterekkel jelölt típusok helyére a kapott konkrét típusok helyettesítődnek be.
  Ettől fogva már példányosíthatjuk a konkrét osztályt.

```
public class Comparer<T>{
  private T    item;
  public  T    getItem(){}
  public void setItem(T item){...}
  public int  compare(T otherItem){...}
}
...
Comparer<String> comparer = new Comparer<>();
comparer.setItem("szöveg");
int compared = comparer.compare("valami");
```

Generikus osztályok

- A generikus osztályok használata is egyfajta absztrakció, de az absztrakt osztályokkal ellentétben itt nem az elvégzendő műveletek megvalósítása az ismeretlen, hanem az adatok típusa (legalább részben), amelyeken a műveleteket végezzük.

```
public class Comparer<T>{
  private T item;
  ...
}

public abstract class GeometricShape{
  public abstract double getArea();
}
```

Generikus osztályok

- A két absztrakciót akár együtt is alkalmazhatjuk.

```
public abstract class ItemProcessor<T> {
  public abstract T getProccessedItem();
  ...
}
```

Gyűjtemények

- A gyűjtemény egy absztrakt adatszerkezet
  - tetszőleges mennyiségű adat csoportosítását végzi
  - az adatok az adott probléma megoldása szempontjából egyformán fontosak
  - az adatokon szabályozott módon lehet műveleteket végezni
- A tárolt adatok általában egyforma típusúak, vagy legalábbis ugyanabból a típusból vannak származtatva
- A tömböket nem tekintjük gyűjteményeknek, mert rögzített mérettel rendelkeznek. Igaz, a gyűjtemények megvalósításához gyakran használunk tömböket.

Gyűjtemények – Példa

```
public class SampleCollection<E> implements Collection<E> {
  @Override public int size(){...}                  // tárolt elemek "száma"
  @Override public boolean isEmpty(){...}             // üres-e?
  @Override public boolean contains(Object o){...} // benne van?
  @Override public Iterator<E> iterator(){...}    // bejárás felsorolója
  @Override public boolean add(E e){...}            // elem hozzávétele
  @Override public boolean remove(Object o){...} // elem eltávolítása
  @Override public void    clear(){...}             // gyűjtemény kiürítése
}
```

Gyűjtemények – Példa

```
public class SampleCollection<E> implements Collection<E> {
  ...
  @Override // a gyűjtemény minden eleme benne van?
  public boolean containsAll(Collection<?> c){...}

  @Override // hozzáadja a gyűjtemény elemeit
  public boolean addAll(Collection<? extends E> c){...}

  @Override // eltávolítja a gyűjtemény elemeit
  public boolean removeAll(Collection<?> c){...}

  @Override // meghagyja a gyűjtemény elemeit
  public boolean retainAll(Collection<?> c){...}

  @Override public Object[] toArray(){...}     // tömbbé konvertálja

  @Override public <T> T[] toArray(T[] a){...} // tömbbé konvertálja
}
```

Gyűjtemények bejárása

- Egy gyűjtemény bejárásakor a gyűjtemény minden elemét sorra vesszük, és minden elemmel elvégezünk egy adott műveletet
- Általában a gyűjtemények nem indexelhetőek, ezért egy úgynevezett *iterátor* segítségével járhatók be
- Megjegyzés: a `Double` a `double` adattípus ,,beburkoló” osztálya, amelyre most azért van szükségünk, mert generikus paraméter csak osztály lehet.

```
Collection<Double> doubles = Arrays.asList(2.72, 3.14, 42.0);
double sum = 0.0;
for (Iterator<Double> it = doubles.iterator(); it.hasNext();){
  Double d = it.next();
  sum += d;
}
System.out.println(sum);
```

Gyűjtemények bejárása

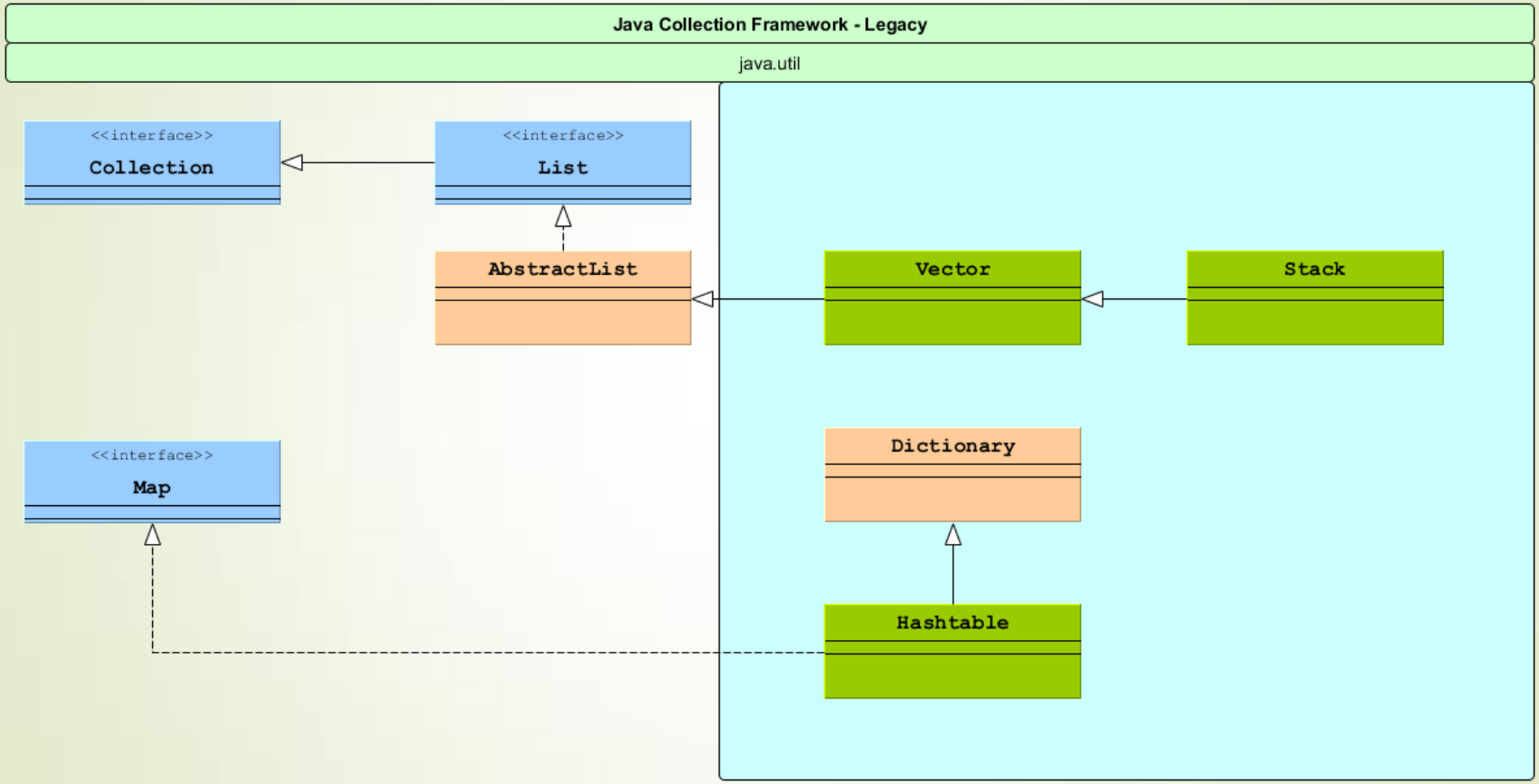
- A gyűjtemények egyszerűbben is bejárhatók, ha a *foreach* ciklust használjuk

```
Collection<Double> doubles = Arrays.asList(2.72, 3.14, 42.0);
double sum = 0.0;
for (Double d : doubles) { sum += d; }
System.out.println(sum);
```

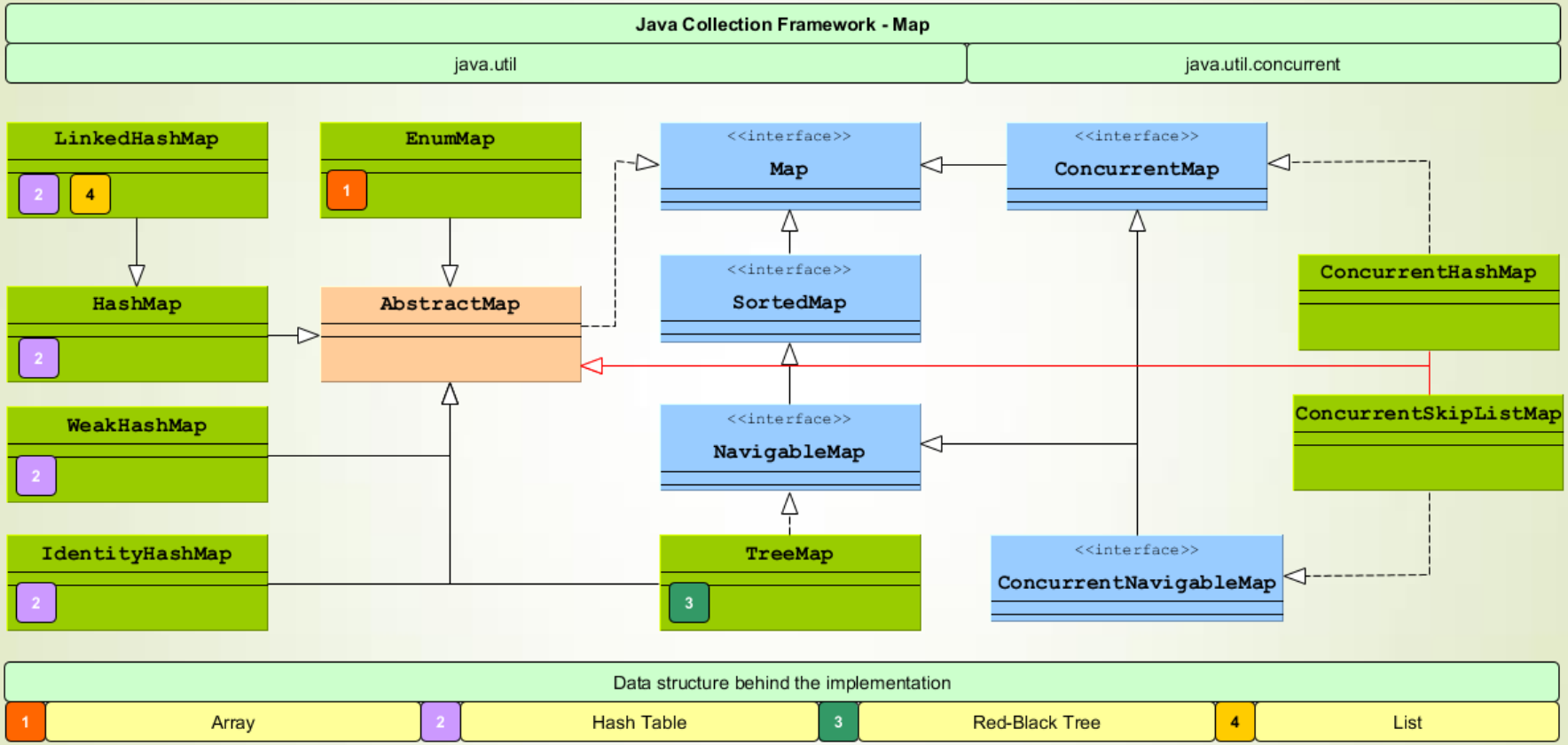
Gyűjtemények megvalósítása

- A `Collection<E>` interfészt, vagy akár valamelyik speciálisabb interfészét kell megvalósítani
- Érdemes a megvalósított gyűjteménynek is generikus osztálynak lennie, hogy tetszőleges típusú adat tárolására alkalmas legyen
- A gyűjtemény műveleteinek absztrakt formái a megvalósítandó interfészben már adottak, így csak a tárolt adatok reprezentációjával és a műveletek függvénytörzseinek meghatározásával kell törődnünk
- Az `AbstractCollection<E>` osztály már tartalmazza a szokásos gyűjteményi viselkedést, így érdemes abból származtatni a gyűjteményünket, és csak a lényegre koncentrálni

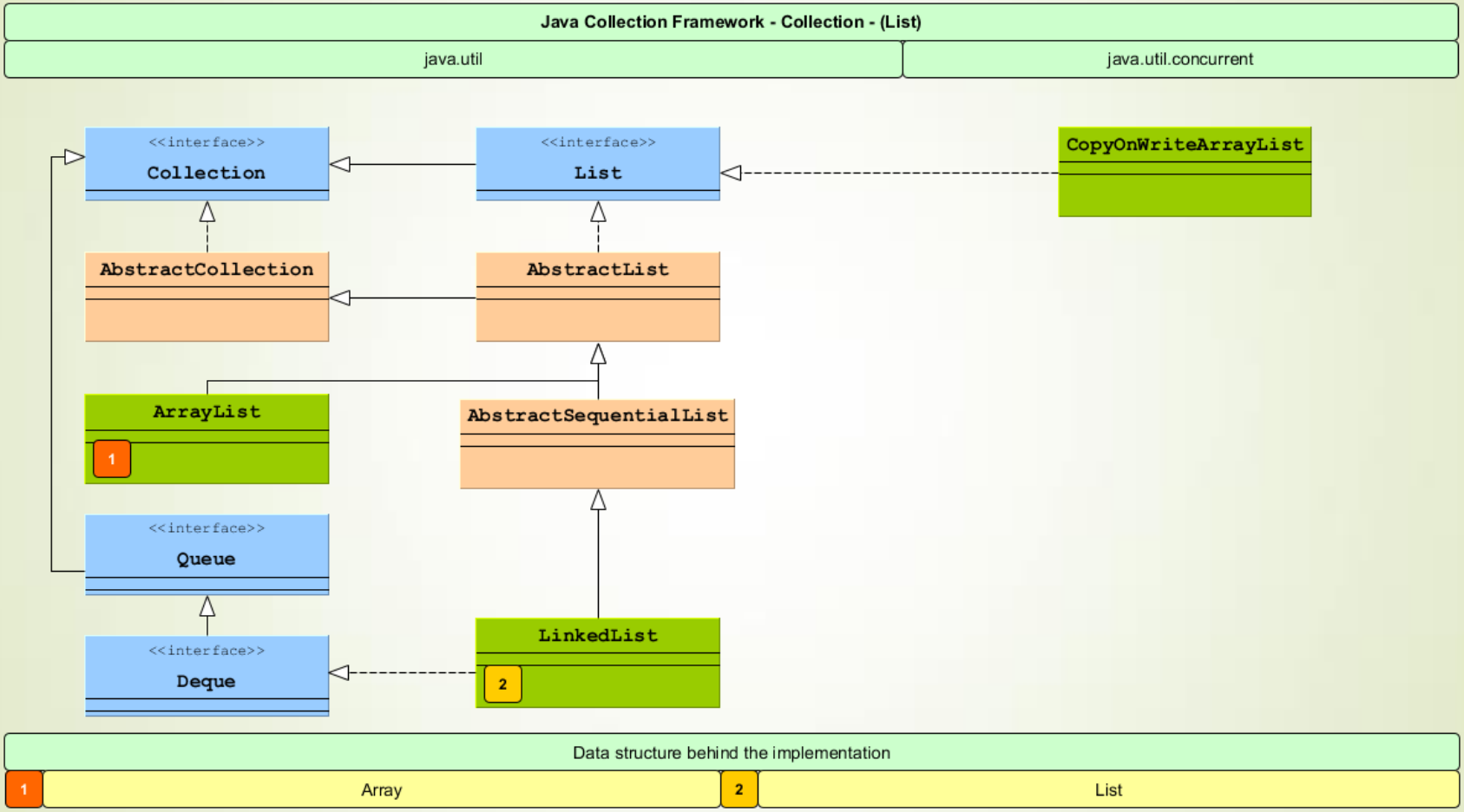
A Java gyűjteményei



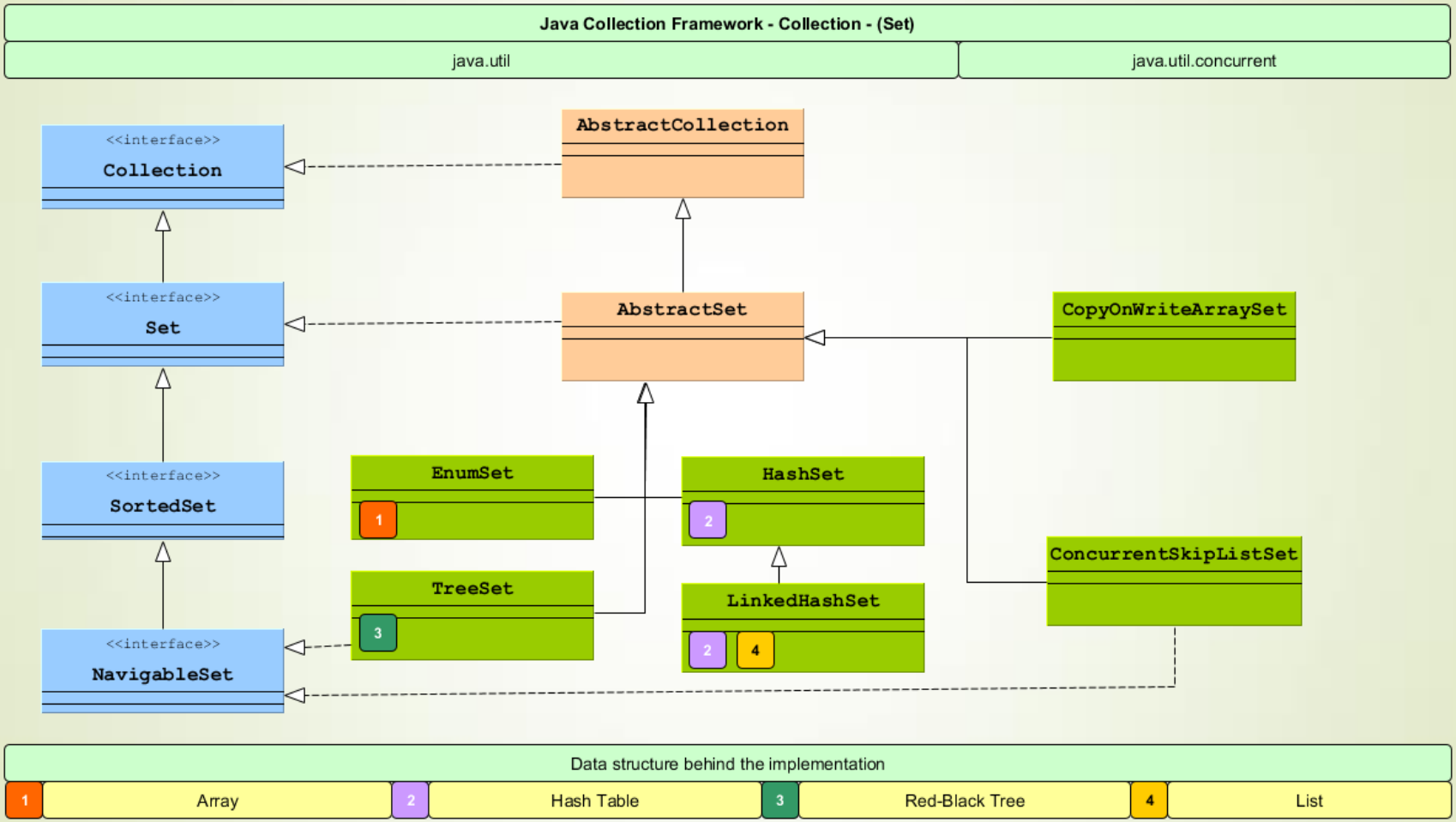
A Java gyűjteményei



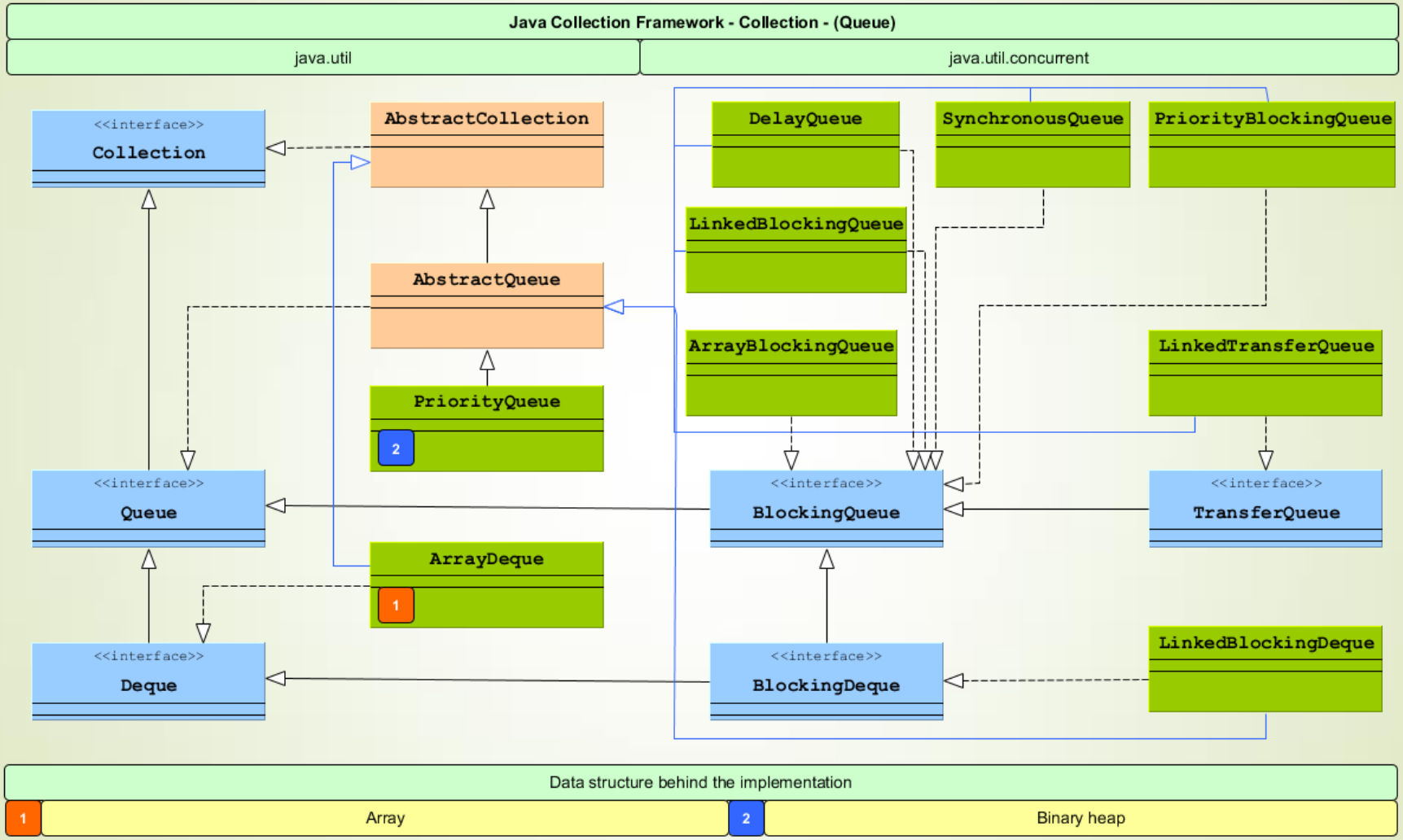
A Java gyűjteményei



A Java gyűjteményei



A Java gyűjteményei



A Java gyűjteményei

Gyűjtemények, mint adatszerkezetek

- A gyűjteményekben tárolt adatok elérésének, módosításának, törlésének ideje nagyban függ a választott adatszerkezettől
  (lásd Algoritmusok és adatszerkezetek tárgy)
- Ökölszabályok kezdőknek:
  - kritikus futási idő alapján választunk adatszerkezetet (pl. keresés és beszúrás);
  - adatok feldolgozási módja szerint választunk adatszerkezetet (pl. rendezés);
  - Kis elemszám esetén kényelmesen dolgozhassuk fel az adatokat;
- speciális feladat esetén készítsünk saját adatszerkezetet (pl. térinformatikai feladatok esetén).

A Java gyűjteményei

Gyűjtemények, mint adatszerkezetek

- A Java gyűjteményeit csoportosíthatjuk a felhasznált adatszerkezet típusa alapján:
  - közvetlen elérésű, indexelhető;
  - láncolt listás;
  - fa adatszerkezetű;
  - hasító függvényt alkalmazó.
- Bizonyos esetekben a fentiek nem eléggé illeszkednek egy adott
- feladathoz, ezért elképzelhető, hogy saját hibrid adatszerkezetet készítünk
  (pl. fa leveleit két irányú láncolt listába fűzzük).

A Java gyűjteményei

Közvetlen elérésű, indexelhető gyűjtemények

- Főbb jellemzők:
  - konstans elem elérési idő,
  - lassú beszúrás,
  - könnyű rendezés

- Gyakran használt Java osztályok:
  - ArrayList,
  - Vector,
  - Stack...

A Java gyűjteményei

Láncolt listás adatszerkezetű gyűjtemények

- Főbb jellemzők:
  - Első/utolsó listaelem azonnal elérhető, a köztes
  - elemek elérési ideje viszont lassú lehet,
  - könnyen bővíthető

- Gyakran használt Java osztályok:
  - Queue,
  - Deque,
  - PriorityQueue,
  - LinkedList...

A Java gyűjteményei

Fa adatszerkezetű gyűjtemények

- Főbb jellemzők:
  - Logaritmikus idejű elem elérés, módosítás és törlés,
  - könnyen bővíthető,
  - az elemek indexelése (n-edik elem kiválasztása) ,,megvalósítható",
  - jól alkalmazhatóak asszociatív tárolókhoz
- Gyakran használt Java osztályok:
  - TreeSet,
  - TreeMap...

A Java gyűjteményei

Hasító függvényt alkalmazó gyűjtemények

- Főbb jellemzők:
  - Gyors elérési, módosítási és törlési lehetőség,
  - kis elemszám / jó hasító függvény esetén nagyon gyors lehet,
  - az elemek nem indexelhetők
  - az elemek rendezése nem lehetséges
  - jól alkalmazhatóak asszociatív tárolókhoz

- Gyakran használt Java osztályok:
  - Hashtable
  - HashSet, LinkedHashSet,
  - HashMap, LinkedHashMap...

Java gyűjtemények használata

Fontos!

- Azon gyűjtemények esetében, ahol az általunk létrehozott típus tárolása a típus elemek értékeinek összehasonlítása alapján történik, kötelező az `equals` metódus megvalósítása az osztályunkban! (Pl.: `Set`, `Map` stb. tárolók esetében)

- Amennyiben a választott gyűjtemény hasítófüggvényt alkalmaz, akkor az `equals` metóduson túl a `hashCode` metódust is meg kell valósítanunk! (Pl.: `HashSet`, `HashMap` stb. esetén)

A Java beépített algoritmusai

- Algoritmusok gyűjteményeken - java.util.Collections

  http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html

- Algoritmusok tömbökön - java.util.Arrays

  http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html