ERHVERVSAKADEMI ÅRHUS

Collections

Java Collections Framework

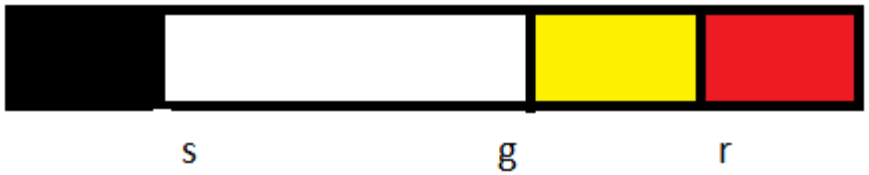
ERHVERVSAKADEMI ÅRHUS

Dagens emner

- Kompleksitet opgaver fra sidst
  - Opgave 1 (bogen)
  - Opgave 2 (størrelsesorden)
  - Opgave 3 (prefixArray)
  - Opgave 4 (regnvejr 1-2 og 3)
  - Opgave 5 (belgisk flag – den tager jeg ☺)
- Collections Framework
  - Collection
  - List
  - Set
  - Map

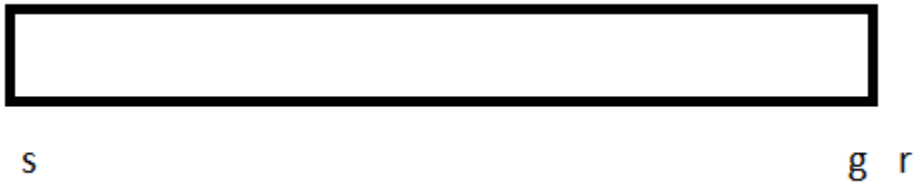
Opgave 5

```
public static void belgiens_flag(char[] belgiens_flag)
    int s = 0;
    int r = belgiens_flag.length;
    int g = belgiens_flag.length - 1;
    while (s <= g) {
        if (belgiens_flag[s] == 'S') {
            s++;
        } else if (belgiens_flag[g] == 'G') {
            g--;
        } else if (belgiens_flag[g] == 'R') {
            r--;
            swap(belgiens_flag, g, r);
            g--;
        } else {
            swap(belgiens_flag, s, g);
            s++;
        }

    }

}
```

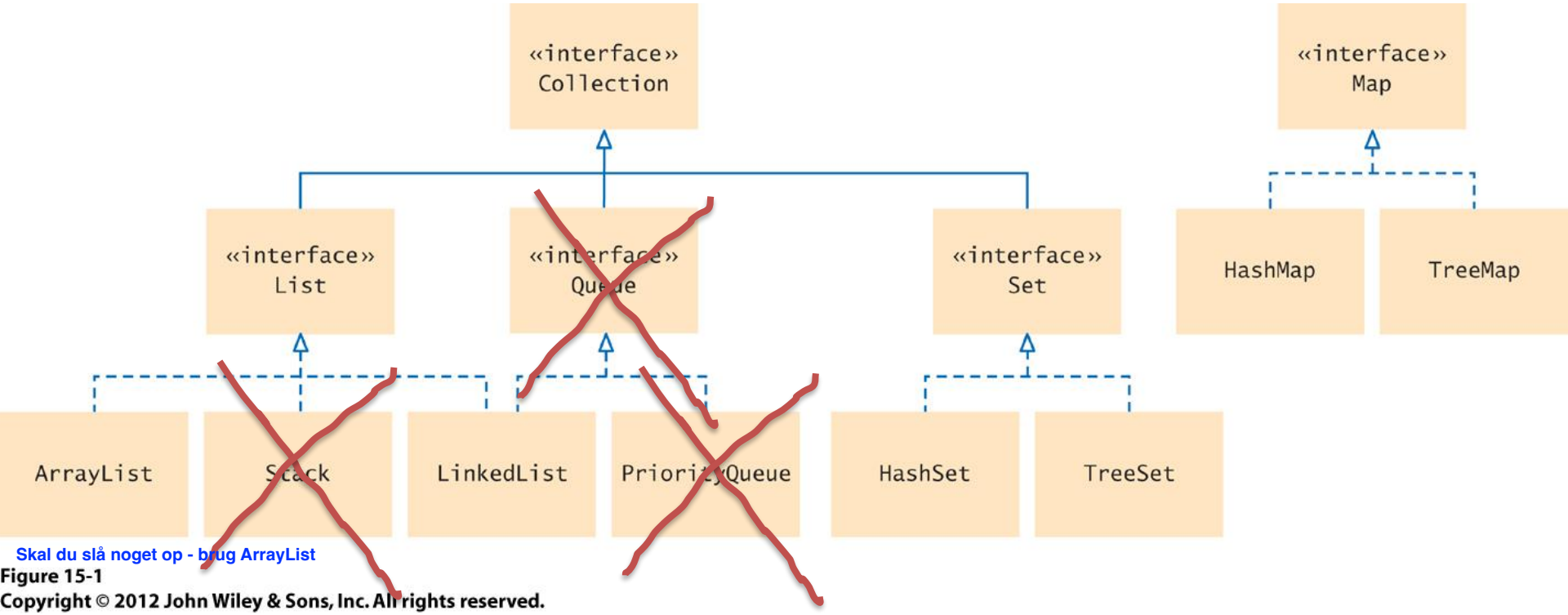


Ved start:

s = 0
g = length-1
r = length



Klassehierarki



Skal du slå noget op - brug ArrayList

Figure 15-1

ERHVERVSAKADEMI ÅRHUS

Collection

Vigtige metoder på alle Collection:

size()	*størrelsen*
toString()	*indholdet som String (kendt fra ArrayList)*
add("Halløj")	*tilføj element*
remove("Halløj")	*fjern element*

```
Collection<String> coll = new ArrayList<>();
for (String s: coll) { System.out.print(s);}
```

ERHVERVSAKADEMI ÅRHUS

Set

Set svarer til en matematisk mængde

- indeholder ikke dubletter
- der er ingen rækkefølge på elementerne
- har samme metoder som Collection
  - add og addAll er implementeret så det samme element ikke indsættes flere gange

ERHVERVSAKADEMI ÅRHUS

List

- I en liste har elementerne en bestemt rækkefølge – elementerne står på en indeks plads i listen
- Indeks starter i 0
- Samme element kan forekomme flere gange i listen
- Har de samme metoder som Collection
- `Har metoder der arbejder med indeks`

get(i)	*Hent element på indeks*
set(i, o)	*Erstat element på indeks i med o*
add(i, o)	*Tilføj element o på indeks i (og skub eksisterende)*
remove(i)	*Fjern element på indeks (og skub de resterende til venstre)*

Map

- I et Map gemmes alle værdier under en nøgle (*key)*
- (key,value) –par gemmes
- key skal altid bruges når en value skal findes
- Har metoderne
  - *put(key,value)* *Gem value under nøgle key*
  - *get(key)* *Hent value der er gemt under nøglen key*
  - *containsKey(key)* *Findes der en value gemt under nøglen key*
  - *keySet()* *Returnerer et Set med alle keys*
  - *values()* *Returnerer en Collection med alle values*

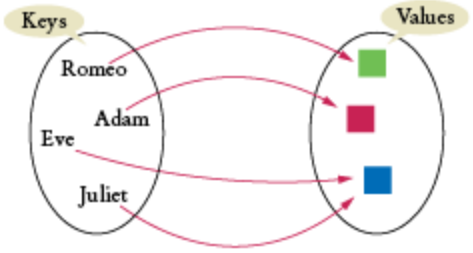
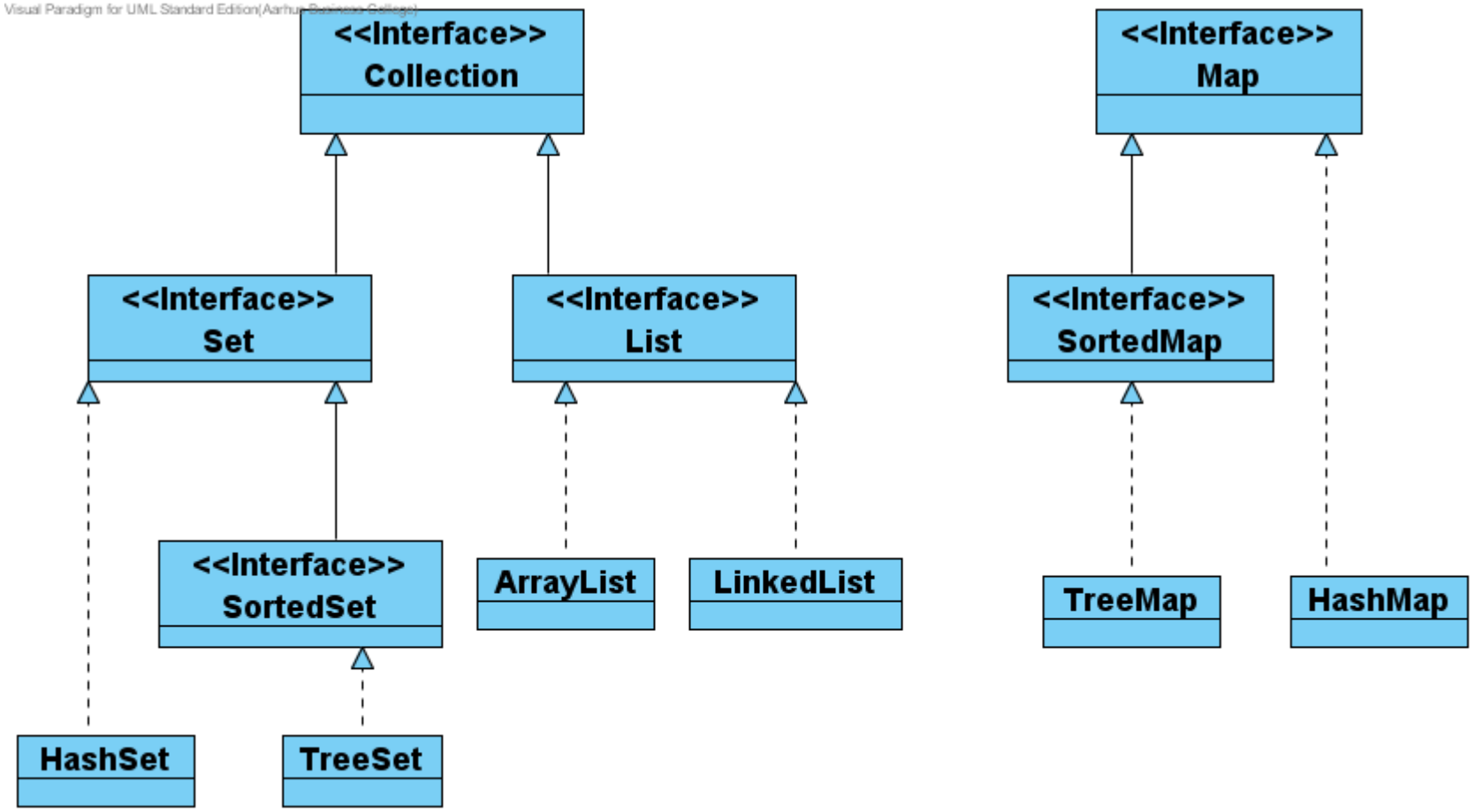


Figure 10 A Map

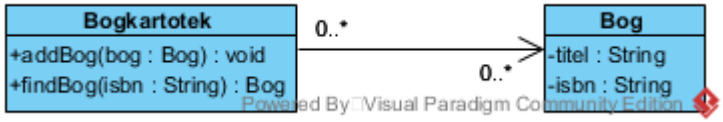
Konkrete klasser i frameworket



Hvorfor dette framework ?

- Ensartet måde at programmere på
- Risikoen for fejl mindre
- Fleksibilitet
  - Let at udskifte en implementation
  - Let at tilføje en ny implementation

Eksempel



Programmer associeringen under anvendelse af:

- ArrayList
- HashSet
- HashMap

ERHVERVSAKADEMI ÅRHUS

Opgaver

Sortering

- TreeSet kræver at objekterne kan sorteres
- TreeMap kræve at nøglerne kan sorteres
- Sortering kan ske ved
  - Comparable (naturlig ordning)
  - Comparator (alternativ ordning)

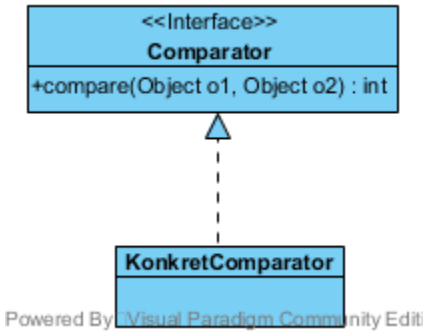
Comparable

Lad klassen implementere Comparable

```
public class Bog implements Comparable<Bog> {
    private String titel;
    private String isbn;

    public Bog(String titel, String isbn) {
        super();
        this.titel = titel;
        this.isbn = isbn;
    }

    @Override
    public int compareTo(Bog inBog) {
        return isbn.compareTo(inBog.getIsbn());
    }

    public String getIsbn() {
        return isbn;
    }
```

Det er den naturlige ordning af Bog objekter

Comparator 1



- Lav en klasse der implementerer Comparator
- Implementer compare metoden
- Lave alternative ordninger på Bog objekter

```
public class BogComparator implements Comparator<Bog>
{
    public int compare(Bog o1, Bog o2) {
        return o1.getTitel().toLowerCase().compareTo(
                o2.getTitel().toLowerCase());
    }
}
```

Comparator 2

```
Bog b1 = new Bog("Java", "4-33");
Bog b2 = new Bog("Pascal", "2-33");
Bog b3 = new Bog("C#", "5-33");
Bog b4 = new Bog("VB", "6-33");

Set<Bog> s = new TreeSet<>();
s.add(b1);
s.add(b2);
s.add(b3);
s.add(b4);
System.out.println(s);
```

Bøgerne udskrives i rækkefølge efter deres naturlige ordning

```
[2-33 Pascal, 4-33 Java, 5-33 C#, 6-33 VB]
```

```
Bog b1 = new Bog("Java", "4-33");
Bog b2 = new Bog("Pascal", "2-33");
Bog b3 = new Bog("C#", "5-33");
Bog b4 = new Bog("VB", "6-33");

Set<Bog> s = new TreeSet<>(new BogComparator());
s.add(b1);
s.add(b2);
s.add(b3);
s.add(b4);
System.out.println(s);
```

Bøgerne udskrives i rækkefølge efter den angivne Comparator

```
[5-33 C#, 4-33 Java, 2-33 Pascal, 6-33 VB]
```

Comparator 3

```
List<Bog> bogliste = new LinkedList<>();
bogliste.add(b1);
bogliste.add(b2);
bogliste.add(b3);
bogliste.add(b4);
Collections.sort(bogliste);
System.out.println(bogliste);


Collections.sort(bogliste, new BogComparator());
System.out.println(bogliste);
```

Bøgerne sorteres i rækkefølge efter deres naturlige ordning

```
[2-33 Pascal, 4-33 Java, 5-33 C#, 6-33 VB]
```

Bøgerne sorteres i rækkefølge efter den angivne Comparator

```
[5-33 C#, 4-33 Java, 2-33 Pascal, 6-33 VB]
```