Algoritmer og datastrukturer

Økt 5 – Generiske metoder

Subklasser

- Student er en person
- Student har studium i tillegg

```java
public static class Person implements Comparable<Person> {
    private final String fornavn;          // personens fornavn
    private final String etternavn;        // personens etternavn

    public Person(String fornavn, String etternavn) {  // konstruktør
        this.fornavn = fornavn;
        this.etternavn = etternavn;
    }

    public String fornavn() {
        return fornavn;
    }       // aksessor

    public String etternavn() {
        return etternavn;
    }   // aksessor
```

```java
public static class Student extends Person {  // Student blir subklasse til Person
    private final Studium studium;      // studentens studium

    public Student(String fornavn, String etternavn, Studium studium) {
        super(fornavn, etternavn);
        this.studium = studium;
    }

    public String toString() {
        return super.toString() + " (" + studium.name() + ")";
    }

    public Studium studium() {
        return studium;
    }

}  // class Student
```

Finn maksimum – forskjellige datatyper

- Hver type krever en egen metode
- Blir mye kopiering av kode
- Finner du en bug i double-versjonen må du fikse den i alle andre versjoner
- Veldig lett å gjøre feil her!

```
/**
 * Finn maksimum
 */
{
    System.out.println("==============================");
    int[] a = {5, 2, 7, 3, 9, 1, 8, 4, 6};
    int k = maks(a);      // posisjonen til den største i a
    System.out.println(Arrays.toString(a));
    System.out.println("Maks int: " + a[k]);

    double[] d = {5.7, 3.14, 7.12, 3.9, 6.5, 7.1, 7.11};
    int l = maks(d);      // posisjonen til den største i d
    System.out.println(Arrays.toString(d));
    System.out.println("Maks double: " + d[l]);

    String[] s = {"Sohil", "Per", "Thanh", "Fatima", "Kari", "Jasmin"};
    int m = maks(s);      // posisjonen til den største i s
    System.out.println(Arrays.toString(s));
    System.out.println("Maks string:" + s[m]);
}
```

Generiske metoder

- Fungerer for flere forskjellige typer!
- Krever at compareTo er implementert i alle objekter du skal sammenlikne
- Dette garanteres av Comparable-interfacet

```java
public static <T extends Comparable<? super T>> int maks(T[] a) {
    System.out.println("maks(comparable)");
    int m = 0;                          // indeks til største verdi
    T maksverdi = a[0];                 // største verdi

    for (int i = 1; i < a.length; i++) {
        if (a[i].compareTo(maksverdi) > 0) {
            maksverdi = a[i];  // største verdi oppdateres
            m = i;             // indeks til største verdi oppdaters
        }
    }
    return m;  // returnerer posisjonen til største verdi
} // maks

/**
 * Finn maksimum med generisk metode
 */
{
    System.out.println("==============================");
    Float[] f = {5.7f, 3.14f, 7.12f, 3.9f, 6.5f, 7.1f, 7.11f};
    int n = maks(f);     // posisjonen til den største i s
    System.out.println(Arrays.toString(f));
    System.out.println("Maks Float: " + f[n]);
}
```

Subklasser

- Generics håndterer dette automatisk!

```
Student[] s = new Student[5];  // en Studenttabell
s[0] = new Student( fornavn: "Kari",  etternavn: "Svendsen", Studium.Data);     // Kari Svendsen
s[1] = new Student( fornavn: "Boris",  etternavn: "Zukanovic", Studium.IT);     // Boris Zukanovic
s[2] = new Student( fornavn: "Ali",  etternavn: "Kahn", Studium.Anvendt);      // Ali Kahn
s[3] = new Student( fornavn: "Azra",  etternavn: "Zukanovic", Studium.IT);     // Azra Zukanovic
s[4] = new Student( fornavn: "Kari",  etternavn: "Pettersen", Studium.Data);   // Kari Pettersen

System.out.println("Før sortering: " + Arrays.toString(s));
innsettingsSortering(s);                  // generisk sortering
System.out.println("Etter sortering: " + Arrays.toString(s));
```

Generiske metoder

```
public static int maks(int[] a) {

public static <T extends Comparable<? super T>> int maks(T[] a) {
```

- Public – offentlig metode (alle kan kalle den)
- Static – denne metoden kan kalles direkte (trenger ikke et objekt)
- T - datatypen vi jobber med, f.eks. int, double, String, etc.
- T extends Comparable<? super T> – Vi krever at T (eller en superklasse av T) implementerer Comparable-interfacet som har funksjonen compareTo
- int – vi skal returnere et heltall fra denne metoden
- T[] a – vi tar inn et array av T'er (for eksempel int[], String[], …)

Studenter har fornavn – kan sammenliknes

- Her er sammenlikneren for Person, men Student er en subklasse av Person!

```
/**
 * Implementering av sammenlikner-interfacet som ser på fornavn av person
 */
public static class FornavnSammenlikner implements Sammenlikner<Person> {
    public int sammenlikn(Person p1, Person p2) {        // to personer
        return p1.fornavn().compareTo(p2.fornavn());  // sammenligner fornavn
    }
}
```

Lambda-funksjoner

- En måte å skrive en lokal funksjon

```
Sammenlikner<Student> c = (s1,s2) -> {
    int cmp = s1.studium().compareTo(s2.studium());
    return (cmp != 0) ? cmp : s1.compareTo(s2);
};

Student[] s = new Student[5];                                    // en studenttabell
s[0] = new Student( fornavn: "Kari", etternavn: "Svendsen", Studium.Data);      // Kari Svendsen
s[1] = new Student( fornavn: "Boris", etternavn: "Zukanovic", Studium.IT);      // Boris Zukanovic
s[2] = new Student( fornavn: "Ali", etternavn: "Kahn", Studium.Anvendt);        // Ali Kahn
s[3] = new Student( fornavn: "Azra", etternavn: "Zukanovic", Studium.IT);       // Azra Zukanovic
s[4] = new Student( fornavn: "Kari", etternavn: "Pettersen", Studium.Data);     // Kari Pettersen

System.out.println("Før sortering: " + Arrays.toString(s));
innsettingsSortering(s, c);                    // se Programkode 1.4.6 b)
System.out.println("Etter sortering: " + Arrays.toString(s));
```

Leksikografiske sammenlikninger

- Sammenlikn først studium, så fornavn, så etternavn

```
System.out.println("==============================");
Sammenlikner<Student> c = (s1, s2) -> {
    int k = s1.studium().compareTo(s2.studium());
    if (k != 0) return k;     // forskjellige studier
    k = s1.fornavn().compareTo(s2.fornavn());
    if (k != 0) return k;     // forskjellige fornavn
    return s1.etternavn().compareTo(s2.etternavn());
};
```

Sammenlikn case insensitivt

```
System.out.println("============================");
String[] s = {"OLE","Per","Kari","PER","Ole","kari","per","KARI","ole"};
System.out.println("Før sortering: " + Arrays.toString(s));
innsettingsSortering(s, Sammenlikner.naturligOrden());
System.out.println("Etter sortering (naturlig): " + Arrays.toString(s));
innsettingsSortering(s, (x,y) -> x.compareToIgnoreCase(y));
System.out.println("Etter sortering (ignore case): " + Arrays.toString(s));
```