3 Interface, exceptions, generik, iteratorer

U 18. Ett interface innehåller bara metodrubriker utan implementeringar (publika, abstrakta metoder). Man får också ha konstanter (attribut deklarerade `static final`). Ett interface utgör ett kontrakt mellan klasser som implementerar det och dess användare. En klass som implementerar ett interface måste implementera alla de abstrakta metoderna i interfacet.

U 19. Både abstrakta klasser och interface innehåller (oftast) abstrakta metoder. Man kan inte skapa objekt av interface eller av abstrakta klasser.

Ett interface innehåller bara metodrubriker, dvs. abstrakta metoder och ev. konstanter. En abstrakt klass kan (precis som en ”vanlig” klass) innehålla attribut, konstruktorer och andra metoder.

U 20. a) Lägg till `implements Resizable` i klassrubriken och implementera metoden `downSize`.

```
public class Rectangle implements Resizable {
    private double height;
    private double width;

    public Rectangle(double h, double w) {
        height = h;
        width = w;
    }

    public void downSize(int scaleFactor) {
        height = height / scaleFactor;
        width = width / scaleFactor;
    }

    ...
}
```

b) 1 och 3 är korrekta. Alternativ 2 är fel eftersom man inte kan skapa objekt av interface.

U 21. a)

```
/**
 * Scales down the object with scaleFactor.
 * @param scaleFactor the scale factor which is used to reduce the
 * size of the object
 * @throws IllegalArgumentException if the specified scale factor is <= 0
 */
public void downSize(int scaleFactor) {
    if (scaleFactor <= 0) {
        throw new IllegalArgumentException("The scale factor has the wrong value: "
            + scaleFactor);
    }
    height = height / scaleFactor;
    width = width / scaleFactor;
}
```

b)

```
try {
    r.downSize(n);
} catch (IllegalArgumentException e) {
    System.out.println(e.getMessage());
}
```

U 22.
```
/** Returns true if this collection contains the specified element. */
public boolean contains(Object x) {
    for (int i = 0; i < size; i++) {
        if (theCollection[i].equals(x)) {
            return true;
        }
    }
    return false;
}

/* Creates an new array that is twice the size of the current array and
   copies the content of the current array into the new one. */
private void doubleArray() {
    E[] tmp = (E[]) new Object[2 * theCollection.length];
    for (int i = 0; i < size; i++) {
        tmp[i] = theCollection[i];
    }
    theCollection = tmp;
}
```

Alternativ implementering av `doubleArray`:

```
private void doubleArray() {
    E[] tmp = (E[]) new Object[2 * theCollection.length];
    System.arraycopy(theCollection, 0, tmp, 0, size);
    theCollection = tmp;
}
```

U 23. Interfacet `Iterable<E>` har en enda metod: `Iterator<E> iterator()`. Klasser som implementerar detta interface måste alltså ha en operation som returnerar ett objekt av en klass som implementerar interfacet `Iterator<E>`. Klasser som implementerar `Iterable<E>` representerar olika slags samlingar av element och operationen `iterator()` ger användare av klassen tillgång till en iterator för elementen, d.v.s. användaren får möjlighet att traversera samtliga element i samlingen. Det objekt som returneras måste vara av en klass som implementerar interfacet `Iterator<E>` som innehåller operationerna `hasNext()` och `next()`. Det är genom anrop av dessa operationer som traverseringen kan genomföras.

U 24. a)
```
Iterator<Person> itr = coll.iterator();
while (itr.hasNext()) {
    System.out.println(itr.next());
}
```

b)
```
for (Person p : coll) {
    System.out.println(p);
}
```

U 25. Lägg till metoden `iterator` som ska skapa och returnera ett iterator-objekt. Iteratorklassen (en klass som implementerar interfacet `iterator` måste också skrivas. Den placeras lämpligen som en inre klass inuti samlingsklassen.

```
public Iterator<E> iterator() {
    return new ArrayIterator();
}
```

```
private class ArrayIterator implements Iterator<E> {
    private int pos;

    private ArrayIterator() {
        pos = 0;
    }

    public boolean hasNext() {
        return pos < size;
    }

    public E next() {
        if (hasNext()) {
            E item = theCollection[pos];
            pos++;
            return item;
        } else {
            throw new NoSuchElementException();
        }
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```