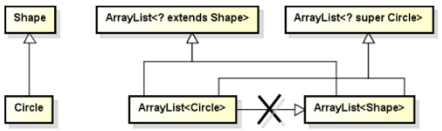
PR2 – Formular für Lesenotizen

SS2021

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	08.05.21

6.10 Generics mit Wildcard <? Extends/super ...>



Übliche Anwendung für <? extends ...>: Objekte aus Quell-Liste lesen:

```
public static void readShapes(ArrayList<? extends Shape> src){
  for (Shape s: src) {
    /* do sth with s */
  }
}
```

```
ArrayList<Shape>
ArrayList<Circle>
ArrayList<Rectangle>
…
```

Übliche Anwendung für <? super ...>: Objekte in Ziel-Liste schreiben.

```
public static void writeCircles(ArrayList<? super Circle> dest) {
  for (int i=1; i<=10; i++) {
    dest.add(new Circle(i));
}}
```

- **Sicher**. Unpassende Übergabe `ArrayList<Integer>` wird vom Compiler verhindert.
- **Flexibel**. Man kann sortenreine `ArrayList<Circle>` übergeben, und auch allgemeine `ArrayList<Shape>`.

- Für `ArrayList<? extends Object>` schreibt man kurz `ArrayList<?>`. `//unterbindet schreiben`

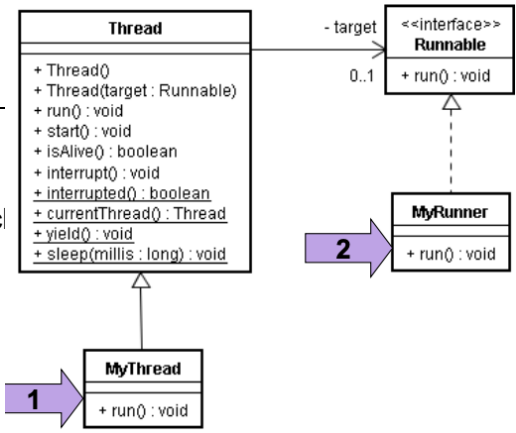
6.11 Finale Klassen und Methoden

- Manchmal möchte man die Vererbung erlauben, aber die Überschreibung einzelner Methoden verbieten.
- Das Schlüsselwort **`final`** vor einer Methodendeklaration drückt aus, dass diese Methode nicht von Subklassen überschrieben werden darf.

```
public final class PinPruefer { }
public final boolean istOk(int pin) { }
```

7. Nebenläufigkeit

Thread: Ausführungsstrang innerhalb eines Prozesses.
„Gleichzeitige" Ausführung ist nur bei mehreren Prozessoren möglich



Alle Threads eines Prozesses greifen auf dieselben Ressourcen desselben Prozesses (Dateien, Fenster, Speicherbereich) zu.

Die Klasse `java.lang.Thread` repräsentiert einen Thread.

1. Subklasse von Thread

```
public class MyThread extends Thread {
  @Override public void run() {
        for (int i=0; i<=3; i++) {
            System.out.println(i);
} } }
```

```
public class ThreadMain {
  public static void main(…) {
    MyThread t = new MyThread();
    t.start();
  }
```

Da die Laufzeit einer Anweisung (gerade bei Ein/Ausgabe-Operationen) nicht immer gleich lang ist, kann es von Ausführung zu Ausführung zu Verschiebungen der Zeitanteile für die einzelnen Threads führen.
→ Man nennt ein solches Laufzeitverhalten **nicht-deterministisch.**

2. Implementierung von Runnable

```
public class MyRunner
                implements Runnable {
  @Override public void run() {
    for (int i=0; i<1000; i++){
      System.out.print('0');
}}
```

Vorteil: Da Mehrfachvererbung von Klassen nicht möglich ist, ist dies manchmal die einzige Möglichkeit, einen Thread zu realisieren.

7.2.1 Threads benennen

```
Thread.currentThread().setName("Hauptthread");
```

```
public class ThreadMain {
 public static void main(String[] args) {
   Thread t= new Thread(new MyRunner());
     t.start();
     for (int i=0; i<1000; i++) {
      System.out.print('#');
}}
Thread t2= new Thread(new Runnable() {
    @Override public void run() {
       for (int i=0; i<1000; i++) { // kuerzer
          System.out.print('0');
} }
new Thread( () -> print('0') ).start(); // Lambda A
```

L.7.4 Einen Thread abbrechen

deprecated Thread `t2.stop();(suspend, resume, destroy) // … NIEMALS EINSETZEN!!!`

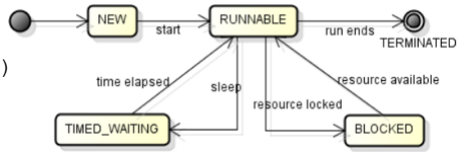
Wie man es macht:

```
t2.interrupt();
if (Thread.interrupted()) {// aufräumen // und run-Methode verlassen;}
```

L.7.5 Einen Thread pausieren lassen

aktives Warten (busy wait) - `for (long j=0; j<20000000; j++); // aktiv warten`

```
try {
    Thread.sleep(50);
} catch (InterruptedException e) {
    break; // wie einen normalen Abbruch behandeln (ausser bei der main)
}
```



L.7.3 Gemeinsam genutzte Daten

Race condition: Die Korrektheit der Ausgabe hängt von der Reihenfolge der Operationen der beteiligten Threads ab.

Kritischer Abschnitt: Abschnitt im Programm, in dem gemeinsame Ressourcen verändert werden und der nicht zeitlich verzahnt von mehreren Threads durchlaufen werden darf.

Zur Analyse kann man **`yield`** mitten im kritischen Abschnitt verwenden, um den **Kontextwechsel** explizit herbei zu führen.

```
Thread.yield(); // Kontextwechsel
```

Gegenseitiger Ausschluss (mutual exclusion): Verfahren zur Lösung von race conditions.
→ kritischen Abschnitt mit dem Schlüsselwort `synchronized`.

Monitor explizit verwenden

synchronized: Kennzeichnung eines kritischen Abschnitts.

```
private static Object lock= new Object();
synchronized (lock) { //kritischer Abschnitt// }
public synchronized void incr() { } // methode als sychhronized markieren!
```

- Wenn eine überflüssigerweise synchronisierte Methode lange Operationen durchführt, blockieren andere Threads unnötigerweise auf Einlass. Der gesamte Programmablauf verzögert sich dadurch.
- Wenn alle Methoden synchronisiert sind, steigt die Gefahr eines Deadlocks.

threadsafe (**threadsicher**): Eine Eigenschaft einer Klasse, die besagt, dass Code der Klasse gleichzeitig von verschiedenen Threads ausgeführt werden kann, ohne dass sich die Threads behindern.

Deadlock (Verklemmung): Zustand, bei dem ein oder mehrere konkurrierende Aktionen darauf warten, dass die jeweils anderen enden, so dass tatsächlich keine Aktion endet.

```
public satic void main(String[] args) {
 long n = einlesen();
 Thread t = new Thread() {
    @Override public void run() {
      rechne(n);
    }};
 t.start();
 warte aufAbbruch(); // Enter..
 if(t.istAlive()){
   System.exit(-1);
}}
public static void einlesen(){ }
public static void rechne( long n){ }
public static void warteAufAbbruch(){ }
```

```
public interface Vehicle {
    public void operate();
}
public interface RailVehicle extends Vehicle {
    public int trackGauge();
}
public interface RoadVehicle extends Vehicle {
    public int numberOfWheels();
}
public interface RoadRailVehicle extends RoadVehicle, RailVehicle {
}

public class CommanderSWX315 implements Vehicle, RoadRailVehicle, RailVehicle,
RoadVehicle  {
    @Override public int numberOfWheels() {return 4;}
    @Override public int trackGauge() {return 1435;}
    @Override public void operate() {System.out.println("…");}
}
public class BMWS1000RR implements Vehicle, RoadVehicle {
    @Override public int numberOfWheels() {return 2; }
    @Override public void operate() {System.out.println("…");}
}
public class AnsaldoFiremaT68 implements Vehicle, RailVehicle{
    @Override public int trackGauge() { return 1435;}
    @Override public void operate() { System.out.println("..");}
}
```