– signatures
– ByteArray
– Stream.sort() which direction it gets sorted
– stream functions, :
– `Function<T,V> Predicate<T> Stream<T> Collection<T>`
– HashCode-methods
– cannot override final methods
– cannot be subclass of final class

Final (Attributes/Parameters)
TODO
Static (Attributes/Methods)
TODO
Private (Attributes/Methods)
TODO
Types

```
long l = 1L; long ll = 0b1l;
float f = 0.0f; long d = 0.0d;
String multiline = """
  Hello, "world"
""";
var ints = new ArrayList<Integer>();
boolean isTrue = 0.1 + 0.1 =≠ 0.2;
```

Variable args

```
long l = 1L; long ll = 0b1l;
static int sum(int... numbers) {
  int sum  = 0;
  for (int i = 0; i < numbers.length; i++) sum +=
numbers[i];
  return sum;
}
```

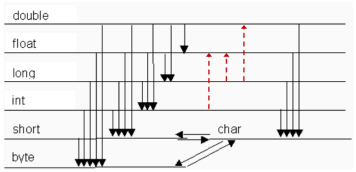
Implicit casting

Rekapitulation: Primitive Datentypen

Konversionen

→ explizit
⇢ implizit (mit evtl. Genauigkeitsverlust)

Sonstige Richtungen implizit



No information loss `int→float`, to larger type `int→long`
Sub->Super is implicit, Super->Sub ClassCastException

Try with

```
try (FileOutputStream output = new
FileOutputStream("filename.txt")) {
  output.write("Hello".getBytes());
} catch (IOException e) {
  System.out.println("Error writing file.");
}
```

Static vs Dynamic types
TODO
Dynamic dispatch
TODO
Equality

```
s.equals(sOther);           // Strings / Objects
Arrays.equals(a1, a2);      // arrays
Arrays.deepEquals(a1, a2);  // nested arrays

class Student extends Person {
  @Override
  public boolean equals(Object obj) {
    if (obj == null) return false;
    if (getClass() ≠ obj.getClass()) return false;
    if (!super.equals(obj)) return false;
    Student other = (Student) obj;
    return getNumber() == other.getNumber();
  }
}
```

String pooling
TODO
Hashing **TODO**
Switch

```
switch (x) {
  case 'a':
    System.out.println("1");
    break;
  default:
    System.out.println("2");
}
int y = switch (x) {
  case 'a' -> 1;
  default -> 2;
}
```

Visibility

public	*all classes*
protected	package + sub-classes
private	only self
(none)	all classes in same package

Packages **TODO**
Initialisation
1) Default-Values ↓
2) Attribute Assignments
3) Initialisation block
4) Constructor

Default Values

Type	*Default*	*Type*	*Default*
boolean	false	char	'\u0000'
byte	0	short	0
int	0	long	0L
float	0.0f	double	0.0d

IO
TODO
Stream API
TODO

```
people
  .stream()
  .filter(p → p.getAge() ⩾ 18)
  .map(p → p.getLastName())
  .sorted()
  .forEach(System.out::println);
```

Lambdas
TODO

```
people.sort(Comparator
  .comparing(Person::getLastName)
  .thenComparing(Person::getFirstName)
  .reversed();
```

Enums
TODO

```
public enum Weekday {
  MONDAY(true), TUESDAY(true), WEDNESDAY(true),
  THURSDAY(true), FRIDAY(true),
  SATURDAY(false), SUNDAY(false);

  private boolean workDay;

  Weekday(boolean workDay) { // private constructor
    this.workDay = workDay;
  }
  public boolean isWorkDay() {
    return workDay;
  }
}
```

Interfaces default methods

```
interface Vehicle {
  default void printModel() {
     System.out.println("Undefined vehicle model");
  }
}
```

Interfaces

```
interface Iterator<T> {
  boolean hasNext();
  T next();
}
class Person implements Comparable<Person> {
  private int age;
  @Override
  public int compareTo(Person other) {
    if (age < other.age) return -1;
    if (age > other.age) return 1;
    return 0;
    // return Integer.compare(age, other.age);
  }
  static int compareByAge(Person p1, Person p2) {
    return Integer.compare(p1.getAge(), p2.getAge());
  }
}
people.sort(Person::compareByAge);
```

Inheritance
Serializable
Compiler Quirks
-> 05_Vererbung_1.pdf …