

Georgiy Shevoroshkin  
 - 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

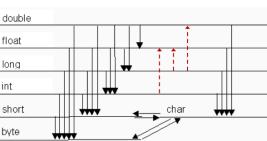
```
long l = 1L; long ll = 0b1l;
float f = 0.0f; long d = 0.0d;
String multiline = """
Hello, "world"
""";
var ints = new ArrayList<Integer>();
boolean isTrue = 0.1 + 0.1 ≠ 0.2;
```

### 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  
**Static vs Dynamic types**

### TODO

### Dynamic dispatch

### TODO

### Equality

```
s.equals(s0ther); // 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;
    Student other = (Student) obj;
    return getNumber() == other.getNumber();
  }
}
```

### String pooling

```
String first = "hello", second = "hello";
System.out.println(first == second); // true
String third = new String("hello");
String fourth = new String("hello");
System.out.println(third == fourth); // false
System.out.println(third.equals(fourth)); // true
String a = "A", b = "B", ab = "AB";
System.out.println(a + b == ab); // false
```

```
final String d = "D", e = "E", de = "DE";
System.out.println(d + e == de); // true
```

### Hashing TODO

### Switch

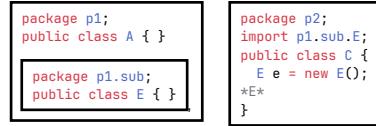
```
switch (x) {
  case 'a':
    System.out.println("1");
    break;
  default:
    System.out.println("2");
}
```

### Visibility

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

### Packages

p1.sub won't be automatically imported in p1.  
 Package name collisions: first gets imported.



```
package p1; public class A { }
package p2; public class E { }
package p1.sub; public class E { }
```

### TODO

### Anonymous Classes 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;
}
```

### Overloading

Gets statically chosen by compiler? But Errors happen at runtime? wtf java?

```
void print(int i, double j) { } // 1
void print(double i, int j) { } // 2
void print(double i, double j) { } // 3
```

```
print(1.0, 2.0); // 3
print(1, 2); // error: reference to print is ambiguous
print(1.0, 2); // 2
```

### TODO

### Overriding

Dynamically chosen (Dynamic dispatch / Virtual call)

**TODO:** Dynamischer Typ des Objektes entscheidet, welche Methode aufgerufen wird

Error: Cannot override the final method...

Error: Cannot be subclass of final class...

### Hiding

```
super.description = ((Vehicle)this).description
super.super // doesn't exist, use v
((SuperSuperClass)this).variable
```

### Abstract classes

```
public abstract class Vehicle {
  private int speed;
  public abstract void drive();
  public void accelerate(int acc) {
    this.speed += acc;
  }
}
public class Car extends Vehicle {
  public void drive() { }
  @Override
  public void accelerate (int acc) { }
}
```

### Interfaces default methods

```
interface Vehicle {
  default void printModel() {
    System.out.println("Undefined vehicle model");
  }
}
public class Qwer {
  @Override
  public void print() {
    System.out.println("1");
  }
}
public class Asdf extends Qwer {
  @Override
  public void print() {
    System.out.println("2");
  }
  public void dostuff () { }
}
var x = new Asdf();
x.print(); // 2
((Qwer) x).print(); // 2
((Qwer) x).dostuff(); // cannot find symbol
```

**Statischer Typ:** Gemäss Variablen Deklaration zur Compile-Zeit

**Dynamische Typ:** Effektiver Typ der Instanz zur Laufzeit

### Serializable

### Compiler Quirks

### Iterators

```
Iterator<String> it = stringList.iterator();
while (it.hasNext()) {
  String s = it.next();
  System.out.println(s);
}
```

Mutating Collection whilst iterating over it: ConcurrentModificationException

Set: No duplicates

### Exceptions

ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException

### void test() throws ExceptionA, ExceptionB {

```
  String c = clip("asdf");
  throw new ExceptionB("wack");
}
```

### try with

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

### Serializing

```
class X implements Serializable { }
// Serializing
try (var stream = new ObjectOutputStream(
  new FileOutputStream("s.bin"))) {
  stream.writeObject(new X());
}
// Deserializing
try (var stream = new ObjectInputStream(
  new FileInputStream("s.bin"))) {
  X x = (X) stream.readObject();
}
```

### Comparable

### TODO

```
var l = new ArrayList<Integer>(asList(3,2,4,5,1));
l.sort((a, b) → a > b ? 1 : -1); // =
l.sort((a, b) → a - b); // 1,2,3,4,5
class Person implements Comparable<Person> {
  private final String firstName, lastName;
  @Override
  public int compareTo(Person other) {
    int result = lastName.compareTo(other.lastName);
    if (result == 0)
      result = firstName.compareTo(other.firstName);
    return result;
  }
  List<Person> people = ...;
  Collections.sort(people);
}
class AgeComparator implements Comparator<Person> {
  @Override
  public int compare(Person p1, Person p2) {
    return Integer.compare(p1.getAge(),
    p2.getAge());
  }
}
people.sort(new AgeComparator());
```