

TODO
– Function<T,V> Predicate<T> Stream<T> Collection<T>
Final (Attributes/Parameters)

| Variable | Method | Class |
|----------|---------------|----------------|
| Constant | No overriding | No inheritance |

- 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 |

Types

```
// char utf16 (16bit) = short
// float 32bit, double 64bit
// short * int      => int
// float + int      => float
// int / double     => double
// int + long * float => float
// 1 / 0.0          => Infinity
// 0.0 / 0.0f       => NaN
//
```

```
long l = 1L;      long ll = 0b1L;
float f = 0.0f;   double d = 0.0d;
```

```
12 = '.'; // implicit int/char conversion
0.1 + 0.1 != 0.2; // true
5/2 = 2; // true, int div truncates to 0
NaN = NaN; // false
Integer.MAX_VALUE + 1 = Integer.MIN_VALUE;
```

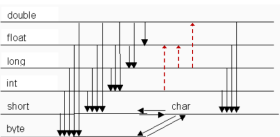
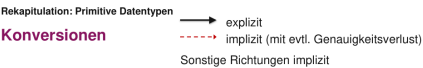
```
String multiline = ""
    Hello, "world"
    """";
"a:b:c".split(":",2).length = 2; // true
```

```
var ints = new ArrayList<Integer>();
int[] jnts = new int[69];
```

```
if (obj instanceof ArrayList<Integer>)
    ((ArrayList<Integer>)obj).add(2);
```

```
public List<String> method(
    BiFunction<Integer, String, List<String>> fn){
    return fn.apply(5, "FooBar");
}
```

Implicit casting



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

```
int[] intarr = new int[] {1, 2, 3, 4, 5};
int[] sub = Arrays.copyOfRange(intarr, 1, 3); // 2,3
```

```
var intlist = new ArrayList<Integer>();
intlist.add(1);
intarr.length; intlist.size();
```

```
// Multiply first to not lose precision
int percent = (int)((filled * 100) / capacity);
```

```
obj.clone();
```

```
Double.POSITIVE_INFINITY; // exists
```

```
Math.min(i,y); Math.max(i,y);
```

Variable args

```
static int sum(int... numbers) {
    int sum = 0;
    for (int i = 0; i < numbers.length; i++)
        sum += numbers[i];
    return sum;
}
```

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;
        if (!super.equals(obj)) return false;
        Student other = (Student) obj;
        return getNumber() == other.getNumber();
    }
}
```

Hashing

Should be added to equals fn's for strict equality

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
```

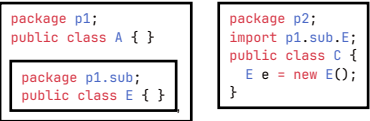
Visibility

| 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 A { }
import p1.A; import p2.*; // OK
import p1.*; import p2.*; // reference to A is ambiguous
import static java.lang.Math.*; // sin, PI
```

IO

```
try (var fr = new FileReader(path)) {
    int input = fr.read();
    while (input ≥ 0) {
```

```
if (input == ';') { /* do something */ }
input = fr.read();
}
```

Enums

```
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;
}
```

```
switch (wd) {
    case MONDAY:
    case TUESDAY:
        System.out.println("First two");
        break;
    default:
        System.out.println("Rest");
}
```

Switch

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

```
int y = switch (x) {
    case 'a' -> 1;
    default -> 2;
}
```

Overloading

Methods with same names but different parameters

Gets statically chosen by compiler

TOD0 with class instances

```
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
print(2.0, (double) 2); // 3
```

Overriding

Methods with same names and signatures
Dynamically chosen (Dynamic dispatch / Virtual call)

Error: Cannot override the final method...

Error: Cannot be subclass of final class...

```
class Fruit {
    void eat(Fruit f) { System.out.println("1"); }
}
class Apple extends Fruit {
    void eat(Fruit f) { System.out.println("2"); }
    void eat(Apple a) { System.out.println("3"); }
}
```

```
Apple a = new Apple();
Fruit fa = new Apple();
Fruit f = new Fruit();
```

```
a.eat(fa); // 2
a.eat(a); // 3
fa.eat(a); // 2
fa.eat(fa); // 2
f.eat(fa); // 1
f.eat(a); // 1
((Fruit) a).eat(fa); // 3
```

```
((Apple) fa).eat(a); // 2
((Apple) f).eat(a); // ClassCastException
```

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 {
    @Override
    public void drive() { }
    @Override
    public void accelerate (int acc) { }
```

Interfaces default methods

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

Interfaces

Cannot have Attributes

```
interface RoadV {
    int MAX_SPEED = 120;
    void drive();
}
```

```
interface WaterV {
    int MAX_SPEED = 80;
    void drive();
}
```

```
class AmphibianMobile implements RoadV, WaterV {
    @Override // because ambiguous
    public void drive() {
        println(RoadV.MAX_SPEED); // MAX_SPEED ambiguous
    }
}
```

```
interface RoadV { String getModel(); }
interface WaterV { int getModel(); }
// Error, because of different return types
class AmphibianMobile implements RoadV, WaterV { }
```

Anonymous Classes

```
var v = new RoadV() {
    @Override
    public void drive() {
        System.out.println("Anon");
    }
}
```

TOD0: mby more interfaces stuff

Inheritance

```
public class Vehicle {
    private int speed;
    public Vehicle(int speed) {
        this.speed = speed;
    }
}
```

```
public class Car extends Vehicle {
    private int doors;
    public Car(int speed, int doors) {
        super(speed);
        this.doors = doors;
    }
}
```

```
Car c = new Car(); // Points to Car
Vehicle v = new Car(); // Points to Car
Object o = new Car(); // Points to Car
// ^static ^dynamic
Car c = (Car) new Vehicle(); // ClassCastException
```

More Inheritance

```
public class Qwer {
    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
```

Static Type: According to var declaration at compiletime

Dynamic Type: Type of the instance at runtime

Iterators

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

Mutating Collection while iterating over it: ConcurrentModificationException

Exceptions

| Error | Exception |
|------------------------------------------------------|---------------------|
| Critical, don't handle | Runtime, handleable |
| OutOfMemoryError, StackOverflowError, AssertionError | IOException |

| Checked | Unchecked |
|-----------------------------------------|-------------------------|
| Must be handled (or throws-declaration) | Not necessary |
| Checked by compiler | Compiler doesn't check |
| Exception, not RuntimeException | RuntimeException, Error |

Child Exception gets caught in catch clause with parent class

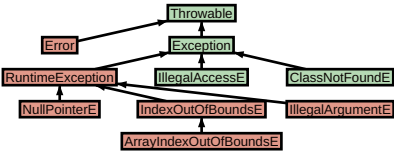
```
void test() throws ExceptionA, ExceptionB {
    String c = clip("asdf");
    throw new ExceptionB("wack");
}
// finally ALWAYS executes, even on unhandled Exc.
try { test() } catch (ExceptionA | ExceptionB e) {
} finally { }
```

```
try { ... } catch(NullPointerException e) {
    throw e; // --leaves blocks→
} catch (Exception e) {
    // above e won't get caught!
}
```

1 / 0; // ArithmeticException div by zero

Unchecked

Checked



Try with

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

Serializing

```

Georgiy Shevoroshkin
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();
}

```

Collection

```

boolean add(E e);           boolean remove(Object o);
boolean equals(Object o);   int hashCode();
int size();                boolean isEmpty();
boolean contains(Object o);

```

```
Set<String> noDup = new HashSet<>();
```

Comparable

```

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;
    }

    static int compareByAge(Person a, Person b) {
        return Integer.compare(a.getAge(), b.getAge());
    }
}

List<Person> people = ...;
Collections.sort(people);
people.sort(Person::compareByAge);

```

```

class AgeComparator implements Comparator<Person> {
    @Override
    public int compare(Person a, Person b) {
        return Integer.compare(a.getAge(), b.getAge());
    }
}

Collections.sort(people, new AgeComparator());
people.sort(new AgeComparator());

```

```

people.sort(Comparator
    .comparing(Person::getAge)
    .thenComparing(Person::getFirstName)
    .reversed())

```

Predicate

```

static void removeAll(Collection<Person> collection,
    Predicate criterion) {
    var it = collection.iterator();
    while (it.hasNext())
        if (criterion.test(it.next()))
            it.remove();
}

```

Lambdas

```

String pattern = readFromConsole();
// vvv not final → Error
while (pattern.length() == 0)
    pattern = readFromConsole();
Utils.removeAll(people, p →
    p.getLastName().contains(pattern));
// local variable ... referenced from a lambda
expression must be final or effectively final

```

Streams

```
import java.util.stream.*;
```

```

people
    .stream()

```

```

    .distinct()
    .filter(p → p.getAge() ≥ 18)
    .skip(5)
    .limit(10)
    .map(p → p.getLastName())
    .sorted()
    .forEach(System.out::println);

```

```

people
    .stream()
    .reduce(0, (acc, cur) → acc + cur.getAge());

```

```

list.stream().mapToInt(Integer::intValue);
list.stream().mapToInt(Integer::parseInt);

```

Terminal operations:

| | |
|--------------------------|-------------|
| min() | max() |
| average() | sum() |
| findAny() | findFirst() |
| forEach(Consumer) | count() |
| forEachOrdered(Consumer) | |