

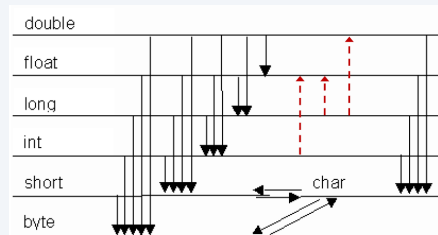
### Java basics

<code>x = a ? b : c</code>	a true? x = b, a false? x = c
<code>0.1 + 0.1 == 0.3</code>	False, workaround: <code>Math.abs(0.1 + 0.1 - 0.2e6) &lt; 0.2e6</code>
<code>a &amp;&amp; b</code>	Only check b if a is true
<code>a    b</code>	Only check b if a is false
<code>0b11001</code>	binary, leading "0b"
<code>0x1e</code>	hexadecimal, leading "0x"
<code>010 != 10</code>	Leading 0 means octal
<code>'A' == 'a'</code>	False (case sensitive)
<code>'A' &lt; 'B'</code>	True

### Java primitive data types

boolean	true, false
char	16 bit, UTF-16
byte	8 bit, -128...127
short	16 bit, -32.768 ... 32.767
int	32 bit, $-2^{31}$ to $+2^{31}-1$
long	64 bit, $-2^{63}$ to $+2^{63}-1$ , <code>long x = 100L;</code>
float	32 bit, <code>float x = 100f;</code>
double	64 bit, <code>double x = 100d;</code>

### Java type casting



red arrows = implicit (probably information loss due to inaccurate data format)  
black arrows = explicit cast (heavy information loss possible --> developer)

### Java interfaces

Methods in interfaces are implicitly public and abstract and can't be private.  
Only constant variables are allowed: `public static final`  
`int HIGHWAY_MAX_SPEED = 60; "public static final"`  
optional.  
Same named methods in basic interface and super interface must have the same return type.  
Same named variables in basic interface and super interface can have different return types.  
Default methods: Basic interface doesn't have to implement default methods. If one method is not overridden, it just takes the default method.

### Java reference types

```
int[] x = new int[10];

int[][] m = new int[2][3];

Arrays.equals(a, b)

Arrays.deepEquals(a, b)

public enum Weekday{ MONDAY, ..., SUNDAY }
```

```
String a = "Progl";

String c = new String("Progl");

a.equals(b)
```

### Java reference types (cont)

```
abstract class Ruler = new String() {
    double toString();
}
```

### Java equals() example

```
@Override
public boolean equals (Object obj) {
    if (obj == null) {
        return false;
    } else if (getClass() != obj.getClass()) {
        return false;
    } else if (!super.equals(obj)) {
        return false;
    }
    // normal Array
    public int[] x = {1, 2, 3};
    // 2 dimensional array
    Student other = (Student) obj;
    // Compare array content
    return regNumber == other.regNumber;
}
// Compare x dimensional array
Enum
```

If equals() is changed to hashCode(), it can't be changed to equals() -> hashCode() == y.hashCode()

### Keywords

Keywords	Compare Strings
public	can be seen by all that imports this package
protected	can be seen by all classes in this package and all subclasses of this
package	can be seen by all classes in this package
private	can be seen only by this class
static	only once for all instances of this class

### Keywords (cont)

**final** can only be defined once and not changed later. Class: no subclasses, Method: no overriding

**static** Means this is a constant  
**final**

### Javadoc

Start with /**	end with */	each line *
@author name	author	class / interface
@version number	version	class / interface
@param name	parameter	method
@return description	returnvalue	method
@throws/@exception type description	potential exception	method
@deprecated description	deprecated (outdated)	method

### Java hashCode() example

```
public int hashCode() {
    return firstName.hashCode() + 31 * surName.hashCode();
}
```

### Java compareTo example

```
class Person implements Comparable<Person> {
    private String firstName,
    lastName;
    // Constructor...
    @Override
    public int compareTo(Person other) {
        int c = compareToString(lastName, other.lastName);
    }
}
```

### Java compareTo example (cont)

```
> if (c != 0) { return c; }
    else { return compareToStrings(firstName,
    other.firstName); }
}
private int compareToStrings(String a, String
b) {
    if (a == null) { return b == null ? 0 : 1; }
    else { return a.compareTo(b); }
}
}
```

### Java collections

```
ArrayList<Object> al = new ArrayList<Object>();
LinkedList<Object> ll = new LinkedList<Object>();
Set<String> s1 = new TreeSet<String>(); or Set<String> s2 = new HashSet<String>();
// TreeSet is always efficient. HashSet = unsorted
```

### Java collections (cont)

```
Map<Integer, Object> m1 = new HashMap<Integer, Object>();
or Map<Integer, Object> m2 = new LinkedHashMap<Integer, Object>();
Iterator<String> it = m1.iterator();
```

### Java inheritance

```
Vehicle v1 = new Car();
Vehicle v = new Vehicle(); Car c = (Car) v;
if (v instanceof Car) { Car c = (Car) v;
    c.add("Test");
}
```

```
Vehicle v = new Vehicle(); Car c = (Car) v;
if (v instanceof Car) { Car c = (Car) v;
    c.add("Test");
}
```

```
super.variable
((SuperSubClass) this).variable
```

**Dynamic dispatch:** Methods: from dynamic type and variables from static type.



## Java Lambdas / Stream API

Collec tio ns.s or t(p eople, (p1, p2

```

Collectio ns.s or t(p eople, (p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15, p16, p17, p18, p19, p20, p21, p22, p23, p24, p25, p26, p27, p28, p29, p30, p31, p32, p33, p34, p35, p36, p37, p38, p39, p40, p41, p42, p43, p44, p45, p46, p47, p48, p49, p50, p51, p52, p53, p54, p55, p56, p57, p58, p59, p60, p61, p62, p63, p64, p65, p66, p67, p68, p69, p70, p71, p72, p73, p74, p75, p76, p77, p78, p79, p80, p81, p82, p83, p84, p85, p86, p87, p88, p89, p90, p91, p92, p93, p94, p95, p96, p97, p98, p99, p100, p101, p102, p103, p104, p105, p106, p107, p108, p109, p110, p111, p112, p113, p114, p115, p116, p117, p118, p119, p120, p121, p122, p123, p124, p125, p126, p127, p128, p129, p130, p131, p132, p133, p134, p135, p136, p137, p138, p139, p140, p141, p142, p143, p144, p145, p146, p147, p148, p149, p150, p151, p152, p153, p154, p155, p156, p157, p158, p159, p160, p161, p162, p163, p164, p165, p166, p167, p168, p169, p170, p171, p172, p173, p174, p175, p176, p177, p178, p179, p180, p181, p182, p183, p184, p185, p186, p187, p188, p189, p190, p191, p192, p193, p194, p195, p196, p197, p198, p199, p200, p201, p202, p203, p204, p205, p206, p207, p208, p209, p210, p211, p212, p213, p214, p215, p216, p217, p218, p219, p220, p221, p222, p223, p224, p225, p226, p227, p228, p229, p230, p231, p232, p233, p234, p235, p236, p237, p238, p239, p240, p241, p242, p243, p244, p245, p246, p247, p248, p249, p250, p251, p252, p253, p254, p255, p256, p257, p258, p259, p260, p261, p262, p263, p264, p265, p266, p267, p268, p269, p270, p271, p272, p273, p274, p275, p276, p277, p278, p279, p280, p281, p282, p283, p284, p285, p286, p287, p288, p289, p290, p291, p292, p293, p294, p295, p296, p297, p298, p299, p300, p301, p302, p303, p304, p305, p306, p307, p308, p309, p310, p311, p312, p313, p314, p315, p316, p317, p318, p319, p320, p321, p322, p323, p324, p325, p326, p327, p328, p329, p330, p331, p332, p333, p334, p335, p336, p337, p338, p339, p340, p341, p342, p343, p344, p345, p346, p347, p348, p349, p350, p351, p352, p353, p354, p355, p356, p357, p358, p359, p360, p361, p362, p363, p364, p365, p366, p367, p368, p369, p370, p371, p372, p373, p374, p375, p376, p377, p378, p379, p380, p381, p382, p383, p384, p385, p386, p387, p388, p389, p390, p391, p392, p393, p394, p395, p396, p397, p398, p399, p400, p401, p402, p403, p404, p405, p406, p407, p408, p409, p410, p411, p412, p413, p414, p415, p416, p417, p418, p419, p420, p421, p422, p423, p424, p425, p426, p427, p428, p429, p430, p431, p432, p433, p434, p435, p436, p437, p438, p439, p440, p441, p442, p443, p444, p445, p446, p447, p448, p449, p450, p451, p452, p453, p454, p455, p456, p457, p458, p459, p460, p461, p462, p463, p464, p465, p466, p467, p468, p469, p470, p471, p472, p473, p474, p475, p476, p477, p478, p479, p480, p481, p482, p483, p484, p485, p486, p487, p488, p489, p490, p491, p492, p493, p494, p495, p496, p497, p498, p499, p500, p501, p502, p503, p504, p505, p506, p507, p508, p509, p510, p511, p512, p513, p514, p515, p516, p517, p518, p519, p520, p521, p522, p523, p524, p525, p526, p527, p528, p529, p530, p531, p532, p533, p534, p535, p536, p537, p538, p539, p540, p541, p542, p543, p544, p545, p546, p547, p548, p549, p550, p551, p552, p553, p554, p555, p556, p557, p558, p559, p560, p561, p562, p563, p564, p565, p566, p567, p568, p569, p570, p571, p572, p573, p574, p575, p576, p577, p578, p579, p580, p581, p582, p583, p584, p585, p586, p587, p588, p589, p590, p591, p592, p593, p594, p595, p596, p597, p598, p599, p600, p601, p602, p603, p604, p605, p606, p607, p608, p609, p610, p611, p612, p613, p614, p615, p616, p617, p618, p619, p620, p621, p622, p623, p624, p625, p626, p627, p628, p629, p630, p631, p632, p633, p634, p635, p636, p637, p638, p639, p640, p641, p642, p643, p644, p645, p646, p647, p648, p649, p650, p651, p652, p653, p654, p655, p656, p657, p658, p659, p660, p661, p662, p663, p664, p665, p666, p667, p668, p669, p670, p671, p672, p673, p674, p675, p676, p677, p678, p679, p680, p681, p682, p683, p684, p685, p686, p687, p688, p689, p690, p691, p692, p693, p694, p695, p696, p697, p698, p699, p700, p701, p702, p703, p704, p705, p706, p707, p708, p709, p710, p711, p712, p713, p714, p715, p716, p717, p718, p719, p720, p721, p722, p723, p724, p725, p726, p727, p728, p729, p730, p731, p732, p733, p734, p735, p736, p737, p738, p739, p740, p741, p742, p743, p744, p745, p746, p747, p748, p749, p750, p751, p752, p753, p754, p755, p756, p757, p758, p759, p760, p761, p762, p763, p764, p765, p766, p767, p768, p769, p770, p771, p772, p773, p774, p775, p776, p777, p778, p779, p780, p781, p782, p783, p784, p785, p786, p787, p788, p789, p790, p791, p792, p793, p794, p795, p796, p797, p798, p799, p800, p801, p802, p803, p804, p805, p806, p807, p808, p809, p810, p811, p812, p813, p814, p815, p816, p817, p818, p819, p820, p821, p822, p823, p824, p825, p826, p827, p828, p829, p830, p831, p832, p833, p834, p835, p836, p837, p838,
```

```
people.st_ream().filter(p -> p.getL
```

## Java Lambdas / Stream API (cont)

```
2 Person[] getAge (peep2egStrAgam(tp;An  
le nqth]);
```

## 2 Possible Stream API operations

```
Pr edicate), map(Fu nct ion), m
AgeToI>=t18Funmapipn> map@D esb Nan
le (Fun ction), sorted(), disti
n ct(), limit(long n), skip(long
ast, Nameq(xlcomand) insaxfa, teven) .
age(), sum()
```

## Comparator & Methodreference

Interface used to compare 2 Objects(before you used lamdas).

Contains the method `compareTo()` which you need to override. Returns positive number if `o1` is bigger than `o2` and negative if opposite. 0 means that they are equal

Instead of a comparator use `methodreference.class::methodName` eg `Person::compareTo`

## Nested class

Use this if a class is only used in another class

No seperate classfile

The inner class can use all members of the outer class.

Instantiation from outside eg `Polygon.Polygon`

Can be declared in a method -> All variables from

```
int getMax Speed() {return 300; } }
```

## Java own exception class

```

private static final long serialVersionUID = 1L;

    public MyException( String
        msg ) {
        super( msg );
    }

    forEach( System.out::println );
}

```

## Java package import conflict order

1. own class (inc, nested class) :: pri ntlr
2. single type imports -> import p2.A;
3. type in own package -> package p1;  
class X
4. Import on demand -> import p2.\*;

## Java regex

1 *	0 to *
1 +	1 to *
1 {2, 5}	min 2 max 5 (11..111)
1 {2, }	min 2 max * (11, 111, ...)
1 {3}	exactly 111
-?1	-1 or 1, "-" is optional
Mo Di	Or
class (this include private members)	any letter from a to z
[a-zA-Z]	any letter from a to z
[a-zA-Z]	any letter from a to z
\s	whitespace
return newSup erC	anything except newli
[^abc]	anything except a, b c
\$	end of string
\d	any digit
\D	not digit
(?< Gro up1 >REG	name capture group -
EX)	r.g rou p("G rou

Example: Check daytime: ( $[0-1] \ ?[0-9]$   
 $[1|2[0-3]]$ ):  $[0-5][0-9]$

### Java regex code example

```
String input =
scanner.nextLine();
Pattern pattern = Pattern.compile ("([0-2]?[0-9]): -
([0-5] [0-9]) ");
Matcher matcher = pattern.matcher (input);
if (matcher.matches()) {
    String hoursPart = matcher.group(1);
    String minute sPart =
matcher.group(2);
    System.out.println (..);
}
```

### Java JUnit

### Java JUnit examples

```
@Test
public void testPrime_2() {
    assertEquals("2 is prime",
utils.isPrime(2));
}
```

### Java generics

**Example:** class Node<T extends Number & Serializable>{ ... } Node<Integer> n = new Node<Integer>(1);  
can add different Interfaces with & to ensure other functionality like serializable

**Wildcard type:** Node<?> undefined Node; undefinedNode = new Node<Integer>(1);  
) and write (.setValue(X)) is allowed

**static variables with generics NOT allowed** eg static Time maxSpeed;

**Generic Method:** public <T> T majority(T x, double p){ if (p > 0.5) { return x; } else { return null; } } **Call:** Double d = test.<Double>majority(1.0, 3.141, new ArrayList<>());

**Rawtype:** like you would insert Object -> you need to down cast the elements. e.g: Node n; // subDepartments) {

### Serializable

Is a marker interface (is empty, just says that this class supports it)

Use it to say the developer that he can serialize objects of this class, which means he can write them in a bytecode and export them. Always serialize all the objects contained in the main object

### Serializable (cont)

Use serialVersionUID to identify your class (no

**Example:** OutputStream fos= new FileOutputStream("file.txt");

**Example:** InputStream fis= new FileInputStream("file.txt");

### Java clone() method

**Example:** public class Department implements Serializable { ... } public Department clone() throws CloneNotSupportedException {

Department d = new Department();  
return d;

**Example:** public class Department implements Serializable { ... } public Department clone() throws CloneNotSupportedException {

Department d = new Department();  
return d;

**Example:** public class Department implements Serializable { ... } public Department clone() throws CloneNotSupportedException {

Department d = new Department();  
return d;

**Example:** public class Department implements Serializable { ... } public Department clone() throws CloneNotSupportedException {

Department d = new Department();  
return d;

<code>assert Equals (expected, actual)</code>	actual «equals» expected
<code>assert Same (expected, actual)</code>	actual== expected (only reference compar- ation)
<code>assert Not Same (expected, actual)</code>	expected != actual (only reference compar- ation)
<code>assert True (condition)</code>	condition
<code>assert False (condition)</code>	!condition
<code>assert Null (value)</code>	value== null
<code>assert Not Null (value)</code>	value!= null
<code>fail()</code>	everytime false
<code>@Test ( timeout= 5000)</code>	set test timeout
<code>@Test ( expected= IllegalArgumentException .class)</code>	expect exception, if exception is thrown, test passes
<code>@Before public void setUp() { ... }</code>	run this before each test
<code>@After public void tearDown() { ... }</code>	run this after each test

