

Secure OOP with Java

Lecture - Unit 02

Claudia Maderthaner <claudia.maderthaner@fh-hagenberg.at>

Data Types

- Java is a **strongly typed** language.
- Every variable must have a declared type.

Types of Data Types

- Primitive Data Types
- Reference Data Types
- Arrays

Primitive Data Types

- Integral Data Types
- Floating-Point Data Types
- The char Type
- The boolean Type

Type	Range		Length	Default	Wrapper
byte	$-2^7 \dots 2^7 - 1$	-128 ... 127	1 Byte	0	Byte
short	$-2^{15} \dots 2^{15} - 1$	- 32 768 ... 32 767	2 Byte	0	Short
int	$-2^{31} \dots 2^{31} - 1$	-2 147 483 648 ... -2 147 483 647	4 Byte	0	Integer
long	$-2^{63} \dots 2^{63} - 1$	-9 223 372 036 854 775 808 ... 9 223 372 036 854 775 807	8 Byte	0L	Long
float	$1.4 \times 10^{-45} \dots 3.4 \times 10^{38}$	7 significant digits (IEEE 754)	4 Byte	0.0f	Float
double	$4.9 \times 10^{-324} \dots 1.7 \times 10^{308}$	15 significant digits (IEEE 754)	8 Byte	0.0d	Double
boolean	true, false		1 Byte	false	Boolean
char	16-bit Unicode (UTF-16)	0x0000 ... 0xFFFF	2 Byte	\u0000	Character

Primitive Literals

```
byte b = 100;  
short s = 10000;  
int i = 100000;
```

Primitive Literals

```
long longExample = 123L;  
double doubleExample1 = 123.4;  
double doubleExample2 = 1.234e2; // 123.4  
float floatExample = 123.4f;
```

Primitive Literals

```
int decimalExample = 26;  
int octalExample = 032;  
int hexadecimalExample = 0x1a;  
int binaryExample = 0b11010;
```


Primitive Literals

```
long groupedNumber = 123_456_783_456L;
```

```
float pi = 3.14_15f;
```

```
long hexWords = 0xcafe_cafe;
```

```
long someByte = 0b0010_0101;
```

The char Type

- Describe individual characters
- Some Unicode characters require **two** char values
- Literal values are enclosed in single quotes.

char Literals

```
char capitalC = 'C';
```

Escape Sequences

Escape Sequence	Name	Unicode Value
\b	backspace	\u0008
\t	tab	\u0009
\n	linefeed	\u000a
\r	carriage return	\u000d
\"	double quote	\u0022
\'	single quote	\u0027
\\	backslash	\u005c

The boolean Type

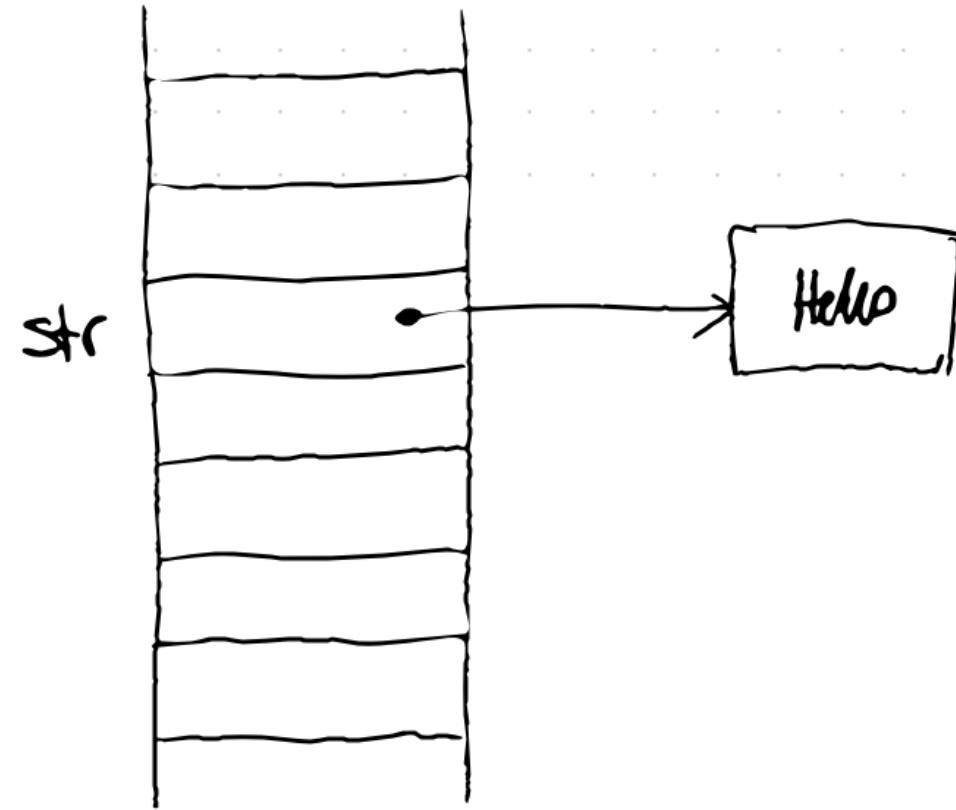
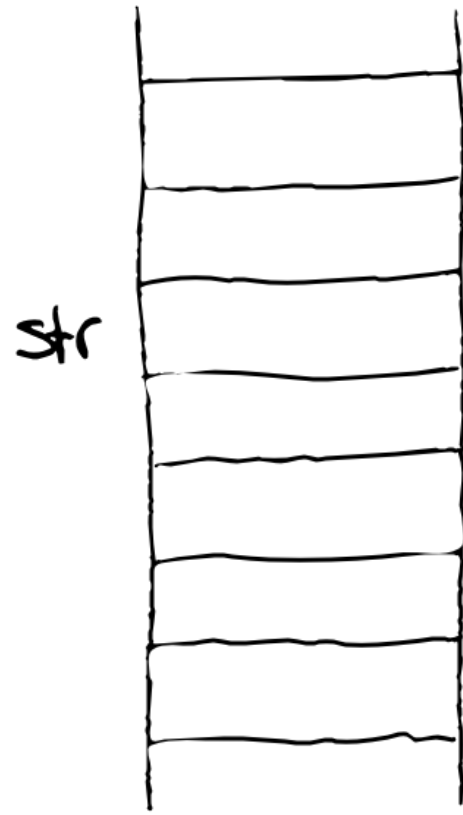
- Two values: `true` and `false`
- Used for evaluating logical conditions
- No conversion between integers and boolean values

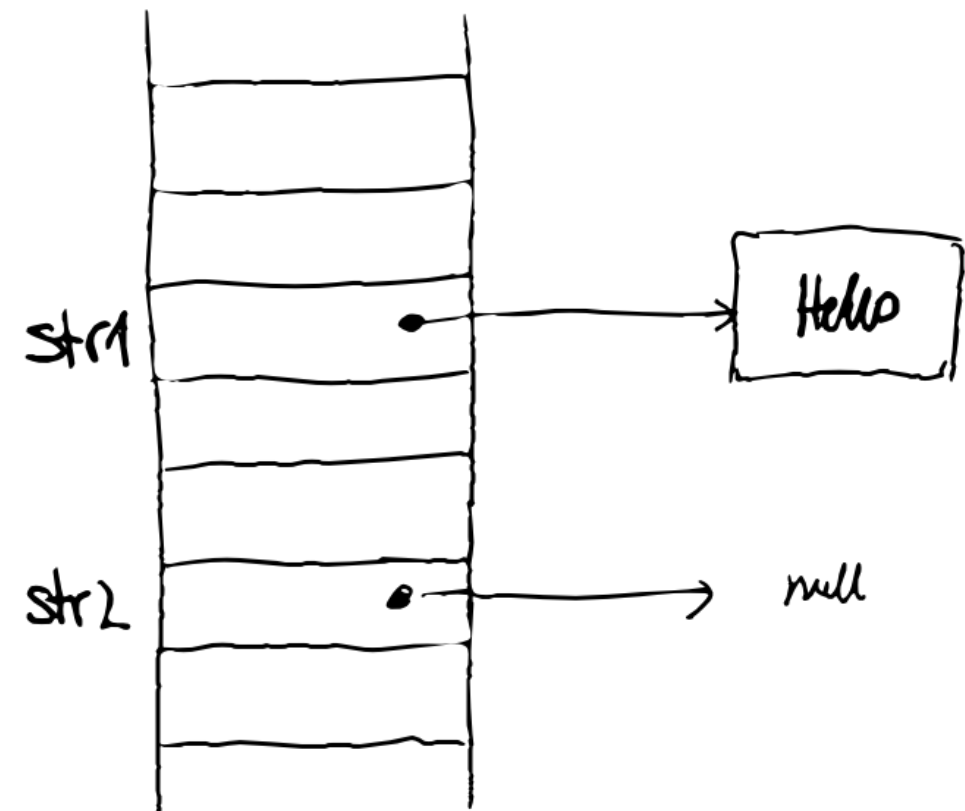
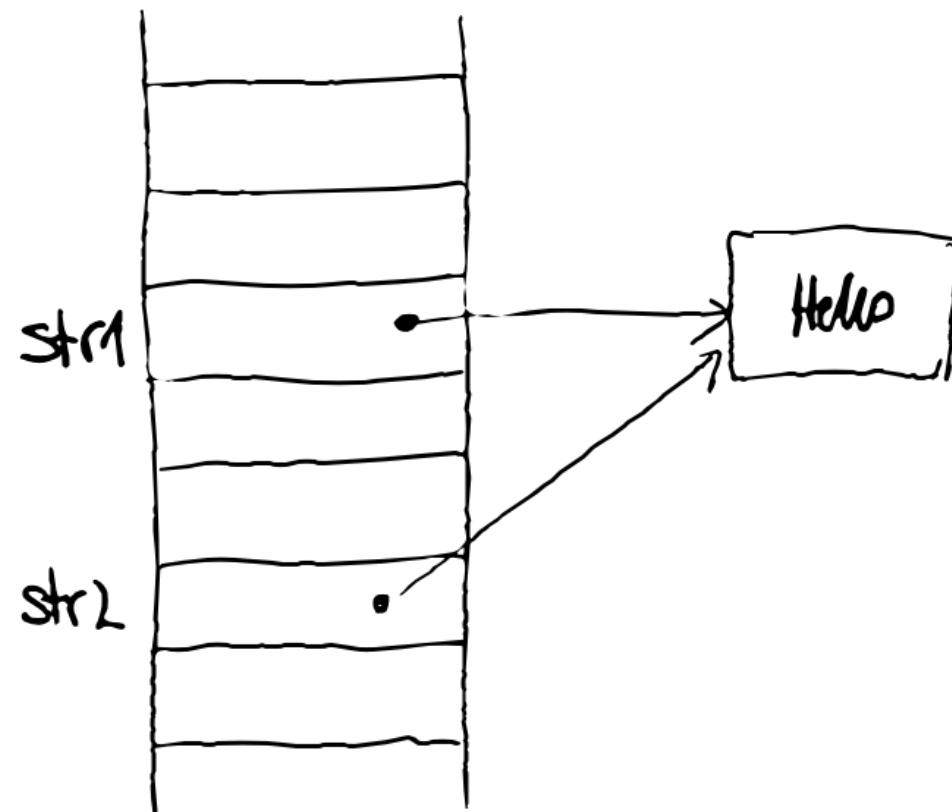
boolean Literals

```
boolean success = true;  
boolean failure = false;
```

Reference Data Types

- Variable holds the reference to an object in memory.
- An object may be referenced from more than one variable.





String

- Predefined class (no primitive type)
- Sequences of Unicode characters

String Literals

```
String line = "Lorem ipsum dolor sit amet, consectetur " +  
              "adipiscing elit. Aenean commodo ligula eget dolor.";
```

String Literals

```
String block = ""  
    Lorem ipsum dolor sit amet, consectetur  
    adipiscing elit. Aenean commodo ligula eget dolor."";
```

Arrays

- Ordered collection of elements
- The type of elements is called the **base type**
- The number of elements it holds is a fixed attribute called `length`

Accessing Array Elements

```
int[] a = new int[10] // => { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }  
for (int i = 0; i < a.length; i++) {  
    a[i] = i;  
}
```

```
a[10] = 10;  
// -> Exception java.lang.ArrayIndexOutOfBoundsException
```

null

- `null` has no type
- Every variable of a reference type may have the value `null`

Typecasts

`long ↔ int ↔ short ↔ byte`

Implicit Typecasts

Widening Conversion

`long ← int ← short ← byte`

Explicit Typecasts

Narrowing Conversion

`long → int → short → byte`

```
int myInteger = 42;  
byte myByte = myInteger; // does not compile  
  
byte myOtherByte = (byte) myInteger;
```

Typecasts Overview

↓ from / → to	boolean	byte	short	char	int	long	float	double
boolean	-	X	X	X	X	X	X	X
byte	X	-	implicit	explicit	implicit	implicit	implicit	implicit
short	X	explicit	-	explicit	implicit	implicit	implicit	implicit
char	X	explicit	explicit	-	implicit	implicit	implicit	implicit
int	X	explicit	explicit	explicit	-	implicit	implicit*	implicit
long	X	explicit	explicit	explicit	explicit	-	implicit*	implicit*
float	X	explicit	explicit	explicit	explicit	explicit	-	implicit
double	X	explicit	explicit	explicit	explicit	explicit	explicit	-

* probable loss of significant digits

Syntax

Variables

Declaration

```
int a;
```

Initialization

```
a = 10;
```

```
int b = 20;
```

Declaring Arrays

```
int[] a; //preferred way of declaration
```

```
int b[];
```

Initializing Arrays

```
int[] a = {1, 2, 4, 5};
```

```
int[] b = new int[3]  
b = new int[] {6, 7, 8}
```


Identifiers

- Name of a language element
- Unlimited length (practically it needs to be less than 64k)
- Cannot begin with a digit
- Cannot be equal to a (reserved) keyword, `null` literal, or `boolean` literal
- Identifiers are case sensitive

Keywords

abstract	assert	boolean	break	byte
case	catch	char	class	const (reserved)
continue	default	do	double	else
enum	exports (restricted)	extends	final	finally
float	for	if	goto (reserved)	implements
import	instanceof	int	interface	long
module (restricted)	native	new	non-sealed (restricted)	open (restricted)
opens (restricted)	package	permits (restricted)	private	protected
provides (restricted)	public	record (restricted)	return	sealed (restricted)
short	static	strictfp	super	switch
synchronized	this	throw	throws	to (restricted)
transient	transitive (restricted)	try	uses (restricted)	var (restricted)
void	volatile	while	with (restricted)	yield (restricted)
_(underscore)				

Restricted keywords are only keywords in a specific context.

Comments

Explanatory text ignored by the compiler

Single Line Comments

```
public final double E = 2.71828182845904523536; // Euler's number
```

Block Comments

```
/*  
 * Return a greeting derived from the given name.  
 */  
return String.format("Hello %s!", name);
```

JavaDoc

```
/**
 * Create a greeting message.
 *
 * @param name the name to be greeted
 * @return a greeting derived from the given name
 */
public String getGreeting(String name) {
    return String.format("Hello %s!", name);
}
```

JavaDoc Tags

Tag	Description	Scope
<code>@author name</code>	author name	class, interface
<code>@version version</code>	version string	class, interface
<code>@since version</code>	API version when item was added	class, interface, field, method
<code>@see reference</code>	associated class name	class, interface, field, method
<code>@param name description</code>	parameter name and description	method
<code>@return description</code>	description of return value	method
<code>@throws class name description</code>	exception name and description	method
<code>@deprecated description</code>	declares item to be obsolete	class, interface, field, method
<code>{@inheritDoc}</code>	copies the description from the overridden method	overriding method
<code>{@link reference}</code>	link to other symbol	class, interface, field, method

workspace - hello-world/src/HelloWorld.java - Eclipse SDK

FileEditSourceRefactorNavigateSearchProjectRunWindowHelp

Package Explorer

hello-world

- JRE System Library [JavaSE-17]
- src
 - (default package)
 - HelloWorld.java

Open Project

Close Project

Build AllCtrl+B

Build Project

Build Working Set

Clean...

Build Automatically

Generate Javadoc...

Properties

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457

1458

1459

1460

1461

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

1472

1473

1474

1475

1476

1477

1478

1479

1480

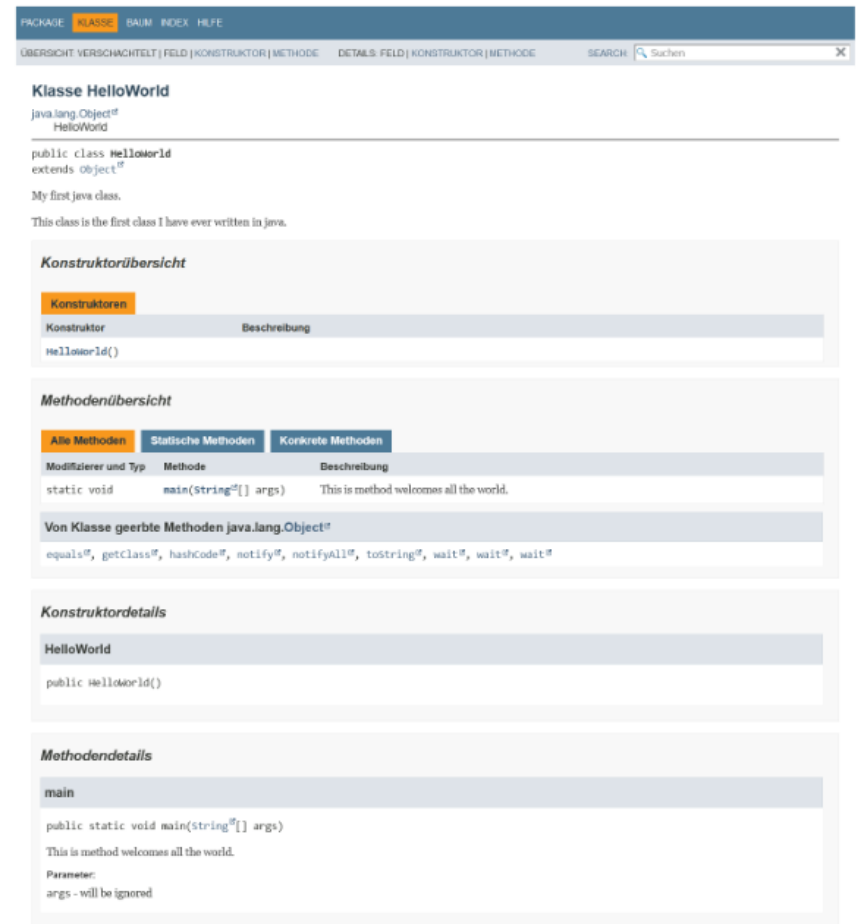
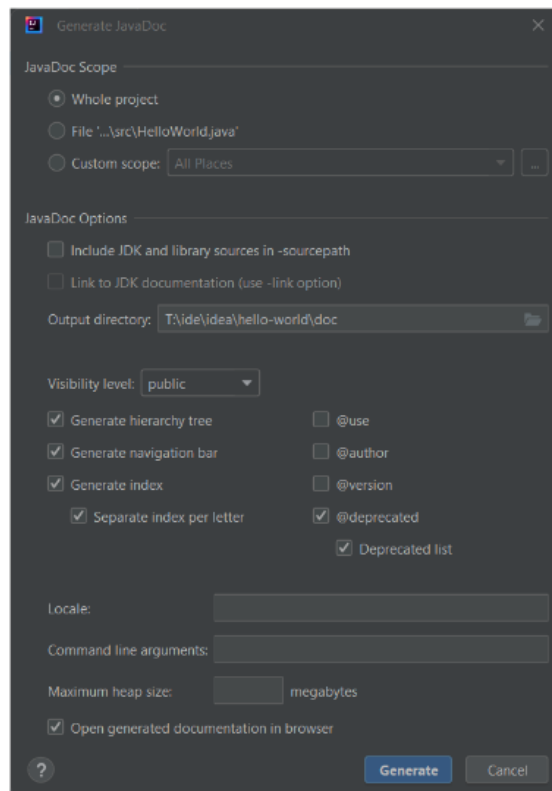
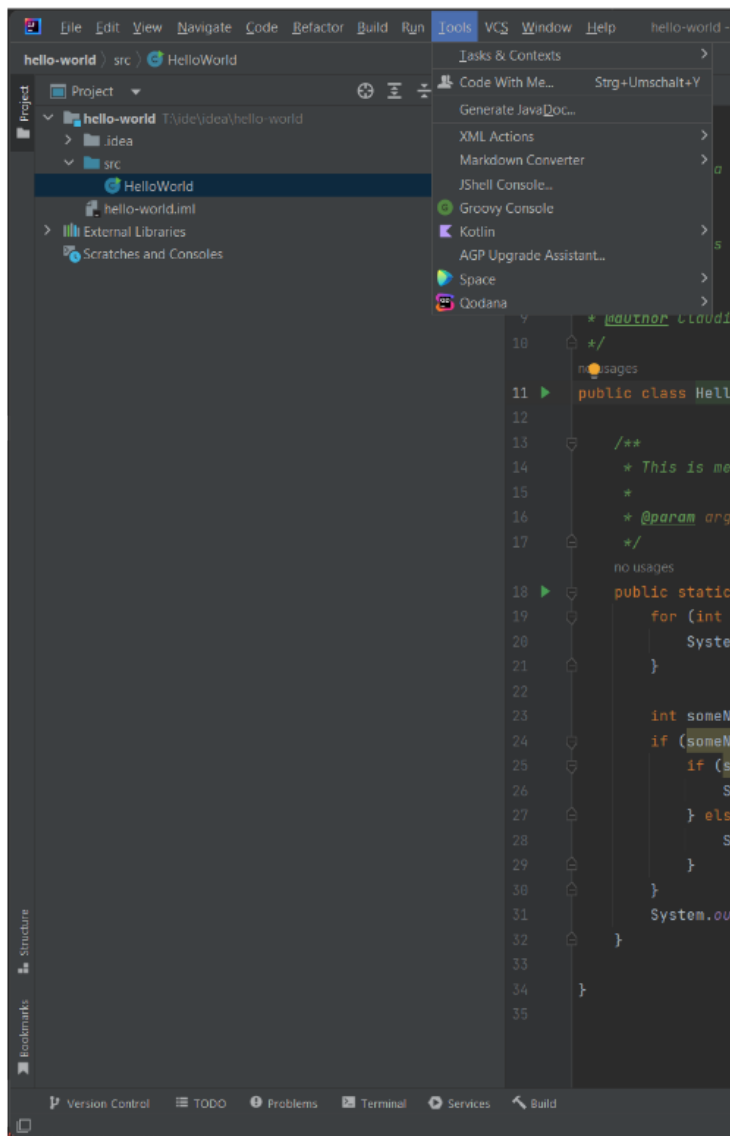
1481

1482

1483

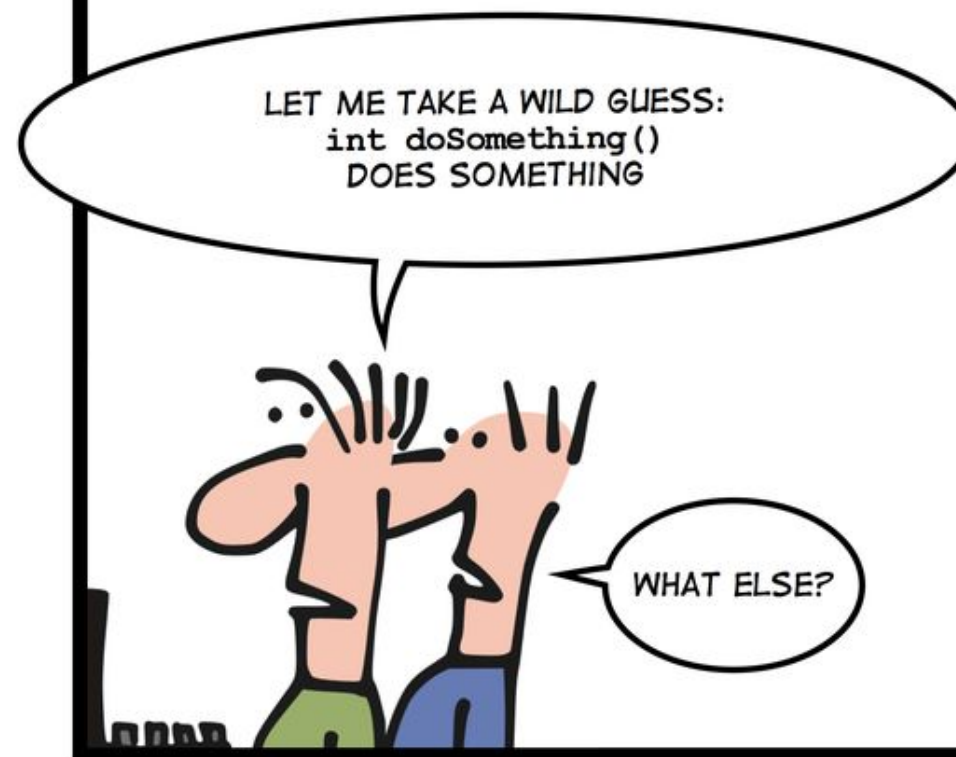
1484

148



SIMPLY EXPLAINED

geek & poke



SELF DOCUMENTING CODE

Imperative Language Features

Expressions

- produce a result, or value, when evaluated
- the value of an expression can be
 - a numeric type
 - a reference type
 - void
- the type of an expression is known at compile time

Operators

Operator	Type		Precedence	Description
++, --	unary	arithmetic	1	increment, decrement
+, -	unary	arithmetic	1	plus, minus
~	unary	integral	1	bitwise complement
!	unary	boolean	1	logical complements
(type)	unary	any	1	cast
*, /, %	binary	arithmetic	2	multiplication, division, remainder
+, -	binary	arithmetic	3	addition, subtraction
+	binary	String	3	String concatenation
<<	binary	integral	4	left shift
>>	binary	integral	4	right shift with sign extension
>>>	binary	integral	4	right shift with no extension
<, ≤, >, ≥	binary	arithmetic	5	numeric comparison
instanceof	binary	object	5	type comparison
=, ≠	binary	primitive	6	equality, inequality of value
=, ≠	binary	object	6	equality, inequality of reference

Operators

Operator	Type		Precedence	Description
&	binary	integral	7	bitwise AND
&	binary	boolean	7	boolean AND
^	binary	integral	8	bitwise XOR
^	binary	boolean	8	boolean XOR
	binary	integral	9	bitwise OR
	binary	boolean	9	boolean OR
&&	binary	boolean	10	conditional AND
	binary	boolean	11	conditional OR
?:	ternary		12	conditional ternary operator
=	binary	any	13	assignment
*, /=, %=, +=	binary	arithmetisch	14	assignment with operation
<<=, >>=, >>> =	binary	integral	14	assignment with operation
&=, ^=, =	binary	integral	14	assignment with operation
+=	binary	String	14	assignment with operation

Arithmetic Conversion

```
byte a = 1;  
byte b = 1;  
  
byte c = a + b; // compile error!  
byte d = (byte) (a + b);
```

Unary minus/plus

```
int a = 1;  
int b = -a;  
int c = -1 * a;  
  
int i = - - - 2 + - + 3;
```


Increment/Decrement

```
int i = 10;  
int j = 20;
```

```
System.out.println(++i); // 11  
System.out.println(--j); // 19  
System.out.println(i); // 11  
System.out.println(j); // 19
```

```
int i = 10;  
int j = 20;
```

```
System.out.println(i++); // 10  
System.out.println(j--); // 20  
System.out.println(i); // 11  
System.out.println(j); // 19
```

Increment/Decrement

```
double d = 12.0;  
System.out.println(--d); // 11.0  
  
double e = 12.345;  
System.out.println(++e); // 13.345
```

Increment/Decrement

```
int i = 1;  
i = ++i;  
System.out.println(i); // 2
```

```
int j = 1;  
j = j++;  
System.out.println(j); // 1
```

Increment/Decrement

```
int a = 10;  
a = a++ + a + a-- - a-- + ++a;
```

Modulo

```
System.out.println(5 / 3); // 1  
System.out.println(5 % 3); // 2
```

```
System.out.println(5 / -3); // -1  
System.out.println(5 % -3); // 2
```

```
System.out.println(-5 / 3); // -1  
System.out.println(-5 % 3); // -2
```

```
System.out.println(-5 / -3); // 1  
System.out.println(-5 % -3); // -2
```

Conditional AND/OR

```
if (obj != null && obj.valid() || obj.obsolete()) {  
    ...  
}
```

Conditional AND/OR

```
int a = 0;  
System.out.println(true || a++ == 0);  
System.out.println(a);
```

```
int b = 0;  
System.out.println(true | b++ == 0);  
System.out.println(b);
```

```
int c = 0;  
System.out.println(false && c++ == 0);  
System.out.println(c);
```

```
int d = 0;  
System.out.println(false & d++ == 0);  
System.out.println(d);
```

bit Operators

a	b	~a	a&b	a b	a^b
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Shift Operators

```
System.out.println(4 << 2);  
System.out.println(-4 << 2);  
  
System.out.println(16 >> 2);  
System.out.println(-16 >> 2);  
  
System.out.println(16 >>> 2);  
System.out.println(-16 >>> 2);
```

Ternary Operator

```
return obj == null ? "" : obj.getName();
```

Assignments

```
int counter = 1;  
counter = 2;  
counter = counter++;  
  
String name = "Herbert";
```

Combined Assignments

```
int a = 10;
```

```
a = a + 1;
```

```
a = a - 3;
```

```
a = a * 5;
```

```
a = a / 4;
```

```
a = a % 2;
```

```
int a = 10;
```

```
a += 1;
```

```
a -= 3;
```

```
a *= 5;
```

```
a /= 4;
```

```
a %= 2;
```

Combined Assignments

```
boolean a = true;  
boolean b = false;
```

```
a = a & b;  
a = a | b;  
a = a ^ b;
```

```
boolean a = true;  
boolean b = false;
```

```
a &= b;  
a |= b;  
a ^= b;
```

Combined Assignments

```
int a = 16;  
  
a = a << 2;  
a = a >> 2;  
a = a >>> 2;
```

```
int a = 16;  
  
a <<= 2;  
a >>= 2;  
a >>>= 2;
```

Statements

Empty Statement

Execution of an empty statement always completes normally.

```
;
```


Blocks and Scopes

```
String question = "What's the time?";  
{  
    String question = "What's the time?"; // compile error  
    String answer = "It's Thursday.";  
    System.out.println(question);  
    System.out.println(answer);  
}  
System.out.println(answer); // compile error
```

if Statements

```
if (condition) {  
    statements;  
}
```

```
if (condition) {  
    statements;  
} else {  
    statements;  
}
```

```
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
}  
else {  
    statements;  
}
```

```
int guess = 73;  
  
if (guess == 83) {  
    System.out.println("You guessed my number!");  
}
```

```
int guess = 73;

if (guess == 83) {
    System.out.println("You guessed my number!");
} else if (guess < 83) {
    System.out.println("Your number is too small!");
} else {
    System.out.println("Your number is too large!");
}
```

```
int someNumber = 0;
if (someNumber >=0)
if (someNumber == 0) System.out.println("first stop");
else System.out.println("second stop");
System.out.println("third stop");
```

Short-Circuit Evaluation

```
int a = 0;
int b = 10;
if (a != 0 && b / a >= 5) {
    System.out.println("Division möglich");
} else {
    System.out.println("Division durch 0 verhindert");
}
```

switch Statement

```
switch (condition) {  
    case OPTION_A:  
        statements;  
        break;  
    case OPTION_B:  
        statements;  
        break;  
    default:  
        statements  
        break;  
}
```

```
switch (dayOfTheWeek) {  
    case 1:  
    case 2:  
    case 3:  
    case 4:  
    case 5:  
        System.out.println("This is a weekday.");  
        break;  
  
    default:  
        System.out.println("This is a weekend day.");  
        break;  
}
```


while Statement

```
while (condition) {  
    statements;  
}
```

```
final int maxValue = 1_000_000;  
int product = 1;  
  
while (product < maxValue) {  
    product *= 2;  
}
```

do-while Statement

```
do {  
    statements;  
} while (condition);
```

```
int input = 0;

Scanner scanner = new Scanner(System.in);
System.out.println("Enter a number (exit with 0)");

do {
    input = scanner.nextInt();
    System.out.println("Your input: " + input);
} while (input != 0);

scanner.close();
```

for Statement

```
for (expression; condition; step) {  
    statements;  
}
```

```
System.out.println("It's the final countdown:");
```

```
for (int i = 10; i >= 0; i--) {  
    System.out.println(i);  
}
```

for-each Statement

```
String[] strings = {"one", "two", "three"};

for (String item: strings) {
    System.out.println(item);
}
```

break/continue

```
int i = 0;
while (true) {
    i++;
    int j = i * 27;
    if (j == 1269) {
        break; // Out of loop
    }
    if (i % 10 != 0) {
        continue; // Top of loop
    }
    System.out.println(i);
}
```


Labels

```
OuterLoop: for (int i = 2;; i++) {  
    for (int j = 2; j < i; j++) {  
        if (i % j == 0) {  
            continue OuterLoop;  
        }  
    }  
    System.out.println(i);  
    if (i == 37) {  
        break OuterLoop;  
    }  
}
```

return Statement

```
public String getGreeting(String name) {  
    return String.format("Hello %s!", name);  
}
```

Unreachable Statements

If a statement cannot be executed because it is **unreachable** leads to a compile-time error.

Contact

Moodle Discussion Board

claudia.maderthaner@fh-hagenberg.at

