

Stile del Codice, Compilazione/Esecuzione avanzata di programmi Java, Programmi con Argomenti, VSCode e Debugging

Programmazione ad Oggetti – Lab02

Docenti: Roberto **Casadei**, Danilo **Pianini**

Tutor: Luca **Deluigi**

C.D.S. Ingegneria e Scienze Informatiche
ALMA MATER STUDIORUM—Università di Bologna, Campus di Cesena

4 aprile 2023



- 1 Stili e Convenzioni per il codice sorgente
- 2 Visual Studio Code (VSCode) e Debugging
- 3 Compilazione ed esecuzione avanzata in Java
- 4 Esecuzioni di programmi java con argomenti
- 5 Laboratorio
 - Appendice: richiami utili per gli esercizi del Lab02



- Rudimenti di programmazione e codifica
- Nozioni di base dei **filesystem**
 - ▶ **percorsi assoluti e relativi**
- Utilizzo del terminale
 - ▶ interazione con il file system attraverso terminale (navigazione; concetto di working directory; etc.)
- **Compilazione ed esecuzione di base** di programmi Java
 - ▶ uso basilare dei comandi `javac` e `java`
 - ▶ distinzione tra file sorgenti (`.java`) e file di classi compilate (`.class`)
 - ▶ concetto di **programma/applicazione** in Java
- Il concetto di **package** in Java
 - ▶ contenitore (organizzato gerarchicamente) di tipi (ad es. classi) che funge da namespace e permette controllo degli accessi ai tipi contenuti



- 1 Stili e Convenzioni per il codice sorgente
- 2 Visual Studio Code (VSCode) e Debugging
- 3 Compilazione ed esecuzione avanzata in Java
- 4 Esecuzioni di programmi java con argomenti
- 5 Laboratorio
 - Appendice: richiami utili per gli esercizi del Lab02



- Il codice sorgente che un programmatore scrive, generalmente è **condiviso** con altre persone (del proprio team, ma anche persone esterne al team o la community)
 - ▶ è importante scrivere software **immediatamente comprensibile**
 - ▶ il fatto che un software “giri” (rispetti i requisiti e/o produca i risultati attesi) non è una sufficiente metrica di qualità
- è importante, fondamentale, adottare uno stile e seguirlo
 - ▶ **chiaro** – facilmente comprensibile
 - ▶ **condiviso** – piuttosto che il “proprio stile”
 - ▶ **consistente** – con regole che non si contraddicono (a vari livelli)



Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. Code for readability.

— John Woods [disputed]

Ogni linguaggio ha le sue prassi...

- Quelle Java di riferimento sono disponibili qui:
 - ▶ <http://bit.ly/java-style-guide>
 - ▶ <http://bit.ly/java-code-conventions>
 - ▶ <http://bit.ly/oracle-java-code-conventions>



Ogni azienda poi è libera di darsi altre regole interne

- Ad esempio:
 - ▶ Google: `http://archive.is/a0Jhz`
 - ▶ Twitter: `http://archive.is/aa1tE`
 - ▶ Mozilla: `http://archive.is/rs3Ns`
- Notare che sono sempre **consistenti!**
 - ▶ E che sono tipicamente restrizioni delle convenzioni, non modifiche!

**Nel corso faremo riferimento alle Java Code Conventions
(con qualche vincolo in più)**



Java Code Conventions (un estratto) I

Usare **sempre** le parentesi (graffe) per if, else, for, while, anche se segue una sola istruzione

- Aumentano la manutenibilità del codice
- È facile che nella fretta si modifichi il codice in modo sbagliato
- È facile che alcuni tool automatici si sbagliano quando “uniscono” pezzi di codice scritti da diverse persone
- Apple iOS soffrì di un grave bug a SSL/TLS causato da questa cattiva pratica <http://archive.is/KQp8E>

Le parentesi graffe vanno sempre “all’egiziana” (Egyptian brackets)

- La graffa che apre va in linea con lo statement di apertura
- La graffa che chiude va in a capo, nella stessa colonna dello statement di apertura

Java Code Conventions (un estratto) II

Naming conventions - **molto importanti!**

- I nomi di package usano sempre e solo lettere minuscole
- Usare sempre camelCase, evitare gli underscore (_)
- I nomi di classe cominciano sempre per maiuscola: SomeClass
- I nomi di campi, metodi e variabili locali iniziano sempre per minuscola: o.someField, o.someMethod()
- I campi `static final` (costanti di classe) sono interamente maiuscoli e possono usare underscore

Come seguire stili e convenzioni? Tutto nelle mani del programmatore?

- Esistono strumenti automatici a supporto, introdotti nelle prossime lezioni...



- 1 Stili e Convenzioni per il codice sorgente
- 2 Visual Studio Code (VSCode) e Debugging
- 3 Compilazione ed esecuzione avanzata in Java
- 4 Esecuzioni di programmi java con argomenti
- 5 Laboratorio
 - Appendice: richiami utili per gli esercizi del Lab02



Visual Studio Code (VSCode)

- **Visual Studio Code (VSCode)** è un editor di codice sorgente leggero, versatile, e multiplatforma
- Estendibile attraverso un ecosistema di **estensioni** per vari linguaggi di programmazione e strumenti
 - ▶ Ne vedremo alcune nei prossimi lab



README.md - oop-lab02 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

- OOPI-LAB02
 - .dist
 - 10-compilation-basics
 - src
 - Calculator.java
 - README.md
 - 11-compilation-with-packages
 - 12-compilation-classpath
 - 13-reference-value
 - 14-program-arguments
 - 15-constructors
 - 16-array

1 # Organizzaz

2

3 #### FASE

4

5 1. Si oss

6 2. Si cre

7 - In l

8 - In l

9 - non

10 3. Si ese

11 - In l

12 - In l

13 java`

14 4. Si verifi

15 5. Si sposti

16 6. Si esegua

17

18 #### FASE 2 - Esecuzione con classpath esplicito

19

20 1. Si posizio

21 2. Da quella

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

1

VSCode: il concetto di **workspace**

- Il **workspace** è
 1. una collezione di *una o più* **cartelle** aperte in una finestra (istanza di VSCode)
 - Tali cartelle sono visualizzate nella vista **Explorer** sulla sinistra
 2. più un insieme di **preferenze, configurazioni, stato**, ed **estensioni** attive memorizzate in una cartella **.vscode/**
- Creazione di un workspace
 - ▶ *File → Open Folder*
- Aggiunta di cartella top-level a un workspace
 - ▶ *File → Add Folder to Workspace..*
- Salvataggio ed apertura di un workspace
 - ▶ *Save → Workspace as... → name.code-workspace*
 - ▶ *File → Open Workspace from File*



Utilizzo di VSCode: alcune note

- Varie scorciatoie da tastiera
 - ▶ *CTRL + SHIFT + P*: **command palette** (a.k.a. “l'unico shortcut che veramente vi serve ricordare”)
 - ▶ *CTRL + SPACE*: intellisense
 - ▶ *CTRL + S*: salvataggio file corrente
 - ▶ *CTRL + PAGE UP/DOWN*: tab sorgente precedente/successivo
- **Apertura di un terminale:** *Terminal* → *New Terminal*
 - ▶ Sarà MOLTO UTILE
- **Visualizzare/installare estensioni:** *File* → *Preferences* → *Extensions* oppure *CTRL + SHIFT + X*



VSCode: debugging di applicazioni Java I

Nota: richiede l'estensione **Debugger for Java**

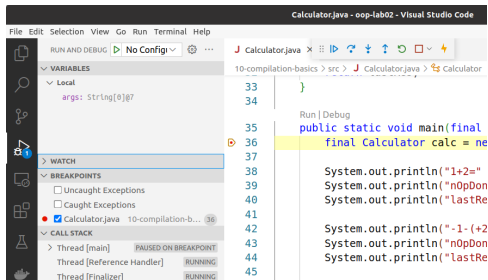
1. Creazione di **breakpoint** (punti di rottura del flusso di controllo)

- Click a sinistra del numero di linea di una riga di codice di interesse

```
34 |  
Run | Debug  
35 | public static void main(final String[] args) {  
• 36 |     final Calculator calc = new Calculator();  
37 |  
38 |     System.out.println("1+2=" + calc.add(n1: 1, n2: 2));
```

2. Esecuzione di un'applicazione Java in “modalità debug”

- Click destro su file java → *Debug Java*



3. Controllo e ispezione del comportamento runtime

- ▶ *Variables*: vista che mostra le variabili in scope e il loro valore
- ▶ *Watch*: permette di valutare espressioni rispetto al contesto d'esecuzione corrente
- ▶ **Step Over** (F10) o anche *Run* → *Step Over*
- ▶ **Step Into** (F11)
- ▶ **Step Out** (SHIFT + F11)
- ▶ **Continue** (F5)



Outline

- 1 Stili e Convenzioni per il codice sorgente
- 2 Visual Studio Code (VSCode) e Debugging
- 3 Compilazione ed esecuzione avanzata in Java**
- 4 Esecuzioni di programmi java con argomenti
- 5 Laboratorio
 - Appendice: richiami utili per gli esercizi del Lab02



Nuova opzione per javac

- Abbiamo già visto come compilare file sorgenti Java (file .java), generando classi in bytecode, che prendono la forma di file .class nella medesima directory
- Tuttavia è uso comune e **buona pratica** nella gestione di progetti articolati, **separare le classi sorgenti dal bytecode**, ad esempio:
 - ▶ cartella src, per i file sorgenti (.java)
 - ▶ cartella bin, contenente le classi compilate (.class)
- Come si fa?

Nuova opzione del comando javac

- -d: consente di specificare la cartella destinazione in cui compilare i file .java
- Si tratta di un'opzione che dovete obbligatoriamente saper usare

Sarà oggetto di valutazione in sede di prova pratica!

Compilazione di più file da qualunque directory verso una qualunque directory

Compilazione in directory arbitrarie

```
javac -d <CARTELLA DESTINAZIONE> <FILE JAVA>
```

- **OVVIAMENTE** vanno sostituite le variabili fra parentesi angolari con le directory che andranno usate.

Compilazione di più file in una singola passata

```
javac -d <CARTELLA DESTINAZIONE> <ELENCO DI FILE JAVA>
```

- **OVVIAMENTE** vanno sostituite le variabili fra parentesi angolari con le directory che andranno usate.

È possibile anche utilizzare la wildcard (*) invece di elencare tutti i file!

- Su Unix si possono usare wildcard in più punti del path, ad esempio `progetti/*/src/*.java` elenca tutti i file con estensione java dentro ciascuna cartella `src` di ciascuna cartella dentro `progetti`



Il classpath in Java I

- Il risultato della compilazione di sorgenti Java sono una o più **classi**
 - ▶ A partire dalla cartella di destinazione (opzione `-d` di `javac`), ogni compilato `.class` sarà creato in un **sottopercorso di cartelle che corrisponde al percorso del package dichiarato per la classe corrispondente**
 - ▶ Ovvero, indipendentemente da dove si trovi un sorgente `C.java` definente una classe `foo.bar.C`, con `javac -d <DEST> path/to/C.java` il compilato sarà creato in `<DEST>/foo/bar/C.class`
- Quando si va ad eseguire (comando `java`), si eseguono **classi**, non files
 - ▶ Infatti la virtual machine si aspetta il nome completo di una classe, il **Fully-Qualified Class Name (FQCN)**, in input
 - NON il percorso al file dov'è scritta
 - NON il percorso al file dov'è compilata



Il classpath in Java II

Come fa la JVM a risolvere le classi?

- Possiede un elenco di percorsi **a partire dai quali** i file compilati possono essere trovati
 - ▶ All'interno di questi percorsi, i file devono essere opportunamente organizzati: **la struttura delle cartelle deve replicare quella dei package**
- Cerca (in ordine) nei suddetti percorsi la classe che gli serve
- I percorsi possono essere directory, file compressi, o indirizzi di rete
- Ad esempio: se si danno i due percorsi `/a/b/c` e `../foo` e si chiede di eseguire il programma definito in `it.unibo.Program`, allora la JVM cercherà di caricare la classe da `/a/b/c/it/unibo/Program.class` o `../foo/it/unibo/Program.class`
- Per approfondire: <http://archive.is/0ziau>

L'insieme *ordinato* dei percorsi prende il nome di **classpath**



Il classpath in Java III

Default classpath

Se non specificato, il classpath di Java include automaticamente:

- I file jar del Java Runtime Environment
 - ▶ `rt.jar` ed altri file importanti
 - ▶ Contengono ad esempio `java.lang.Math`
- La directory corrente

Aggiungere directory al classpath

Possono essere aggiunte directory al classpath

- Si usa l'opzione `-cp` (o, equivalentemente, `-classpath`), seguita da un elenco di percorsi
 - ▶ separati dal simbolo `:` (Unix)
 - ▶ o dal simbolo `;` (Windows)
 - ▶ Per evitare problemi con simboli e percorsi, conviene circondare l'intero classpath con doppi apici (simbolo `"`)

Più cartelle nel classpath

Il classpath non è composto da una sola cartella, ma può contenere più cartelle, oltre a file compressi contenenti bytecode e risorse, come zip o jar (li vedremo in futuro).

Si possono specificare più cartelle utilizzando come separatore il simbolo ":" (per sistemi Unix) oppure ";" (per sistemi Windows)

- `javac -d bin -cp "lib1:lib2:lib3" src/*.java`
 - ▶ Compila tutti i file con estensione java che si trovano nella cartella `src`, mettendo i compilati dentro `bin`. In compilazione, potrà linkare tutte le classi che si trovano nelle cartelle `lib1`, `lib2` e `lib3`: nel caso in cui alcuni sorgenti in `src` stiano usando delle classi definite dentro queste cartelle, la compilazione avrà successo.
 - ▶ Equivalente Windows: `javac -d bin -cp "lib1;lib2;lib3" src/*.java`
- `java -cp "bin:lib1:lib2:lib3" MyClass`
 - ▶ Esegue il main della classe `MyClass`. Cercherà questa classe e tutte quelle collegate all'interno delle cartelle `bin`, `lib1`, `lib2` e `lib3`.
 - ▶ Equivalente Windows: `java -cp "bin;lib1;lib2;lib3" MyClass`

Organizzazione dei sorgenti in presenza di package

È buona norma organizzare i sorgenti in modo da rappresentare su filesystem la struttura dei package. Si noti però che (dato che il compilatore lavora su **file**) questa scelta **non è teoricamente obbligatoria!**

- Lo è di fatto in questo corso, perché le cose van fatte bene
- Lo sarà nel mondo del lavoro, perché è prassi assolutamente comune

Risultato della compilazione

Quando ad essere compilata è una classe dichiarata in un package, il compilatore **riproduce la struttura dei package usando delle directory**

- Dato che l'interprete non lavora con file ma con **classi**, il loro layout sul file system **non può essere modificato!**

Esecuzione

L'esecuzione è identica al caso precedente, si faccia solo attenzione ad usare l'intero nome della classe, che in Java include anche il nome del package!

Uso del classpath in fase di compilazione I

Supponiamo di avere in mano la seguente classe:

```
1 package oop.lab02.math;
2
3 public class UseComplex {
4
5     public static void main(final String[] args) {
6         final ComplexNum c1 = new ComplexNum();
7         c1.build(1, -45);
8         final ComplexNum c2 = new ComplexNum();
9         c2.build(2, 8);
10
11         System.out.println(c1.toStringRep());
12         System.out.println(c2.toStringRep());
13
14         c1.add(c2);
15         System.out.println("c1 new value is: " + c1.toStringRep() + "\n");
16     }
17 }
```



Uso del classpath in fase di compilazione II

Comprensione degli errori

Se provassimo a compilarla da sola, potremmo ottenere degli errori

```
1 src\oop\lab2\math\UseComplex.java:6: error: cannot find symbol
2     ComplexNum c1 = new ComplexNum();
3         ~
4     symbol:   class ComplexNum
5     location: class UseComplex
6 src\oop\lab2\math\UseComplex.java:6: error: cannot find symbol
7     ComplexNum c1 = new ComplexNum();
8         ~
9     symbol:   class ComplexNum
10    location: class UseComplex
11 src\oop\lab2\math\UseComplex.java:8: error: cannot find symbol
12     ComplexNum c2 = new ComplexNum();
13         ~
14 ...
```

- Il compilatore ha bisogno di conoscere la classe `ComplexNum` per poterla linkare e per poter compilare una classe che la riferisce
- Il compilatore cerca nel classpath il bytecode della classe `ComplexNum`
- Come risolviamo?

Uso del classpath in fase di compilazione III

Utilizzo di -cp in fase di compilazione

- Supponiamo di avere solo la versione compilata di `ComplexNum` (ovvero non il sorgente)
 - ▶ Notate che questa è la *norma* quando si usano delle librerie: vengono fornite già compilate!
- Basterà mettere il percorso a partire dal quale `oop/lab02/math/ComplexNum.class` può essere individuata nel classpath di `javac`!
- Supponiamo di avere `UseComplex.java` nel percorso `src/oop/lab02/math/`
- Supponiamo di aver compilato `ComplexNum` con destinazione (di partenza) `lib/`
- Possiamo usare:

```
javac -d bin -cp lib src/oop/lab02/math/UseComplex.java
```

Uso del classpath in fase di compilazione IV

Spiegazione del comando

```
javac -d bin -cp lib src/oop/lab02/math/UseComplex.java
```

- `javac` \Rightarrow Invocazione del compilatore
- `-d bin` \Rightarrow `-d` determina la **destinazione**. Vogliamo compilare dentro la cartella `bin`
- `-cp lib` \Rightarrow `-cp` consente di aggiungere percorsi al **classpath**. Noi vogliamo cercare le classi che ci servono, oltre che nella posizione corrente e nelle librerie java, anche dentro `lib`
- `src/oop/lab02/math/UseComplex.java` \Rightarrow Il *file* che vogliamo compilare



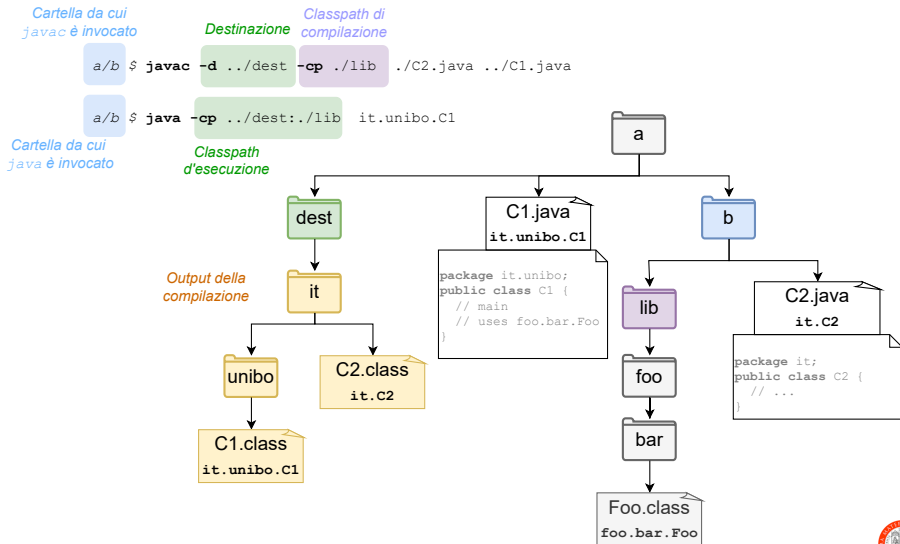
Passare più percorsi al classpath

Avendo come riferimento l'esempio precedente, proviamo ad eseguire.

- Per eseguire correttamente UseComplex dobbiamo dire alla JVM, tramite **-cp**, dove trovare:
 - ▶ ComplexNum
 - ▶ UseComplex
- Si trovano in *due percorsi diversi!*
- Dobbiamo specificare come argomento di **-cp** due percorsi, usando il **separatore**:
 - ▶ : su sistemi UNIX (Linux, Mac OSX, BSD)
 - ▶ ; su sistemi Windows
- Useremo quindi:
 - ▶ `java -cp bin:lib oop.lab02.math.UseComplex (Unix)`
 - ▶ `java -cp bin;lib oop.lab02.math.UseComplex (Windows)`



Esempio con sommario di funzionamento di javac e java



Consiglio finale

Visto che all'esame il loro utilizzo è richiesto, è necessario imparare a **memoria** le opzioni di java e javac?

NO

Entrambi i comandi (e praticamente tutti i comandi Unix) hanno con loro un'opzione che consente di stampare a video un help. Provate

- java -help
- javac -help

Gli help stampano abbondante testo con le relative istruzioni e a me serve una riga, davvero devo imparare a leggere e capire un help?

sì

È molto facile dimenticarsi la sintassi delle opzioni di comandi che non si usano spesso. È molto più facile imparare a destreggiarsi in un help che andare a tentativi o ricordare cose a memoria.



- 1 Stili e Convenzioni per il codice sorgente
- 2 Visual Studio Code (VSCode) e Debugging
- 3 Compilazione ed esecuzione avanzata in Java
- 4 Esecuzioni di programmi java con argomenti**
- 5 Laboratorio
 - Appendice: richiami utili per gli esercizi del Lab02



Passaggio di argomenti ad un programma Java I

- La maggior parte dei comandi supporta degli argomenti
 - ▶ Ad esempio, quando eseguite `javac -d bin MyClass.java` gli argomenti sono:
 1. `-d`
 2. `bin`
 3. `MyClass.java`
- In C, questi vengono passati al metodo `main()` come coppia di `char **` e `int`, rappresentanti rispettivamente un riferimento all'area di memoria dove sono salvati i parametri ed il numero dei suddetti.
- Anche in Java ovviamente è possibile passare degli argomenti ad un programma



Passaggio di argomenti ad un programma Java II

- La gestione è un po' più semplice che in C, grazie al fatto che gli array si portano dietro l'informazione circa la loro dimensione
- E grazie al fatto che la signature del metodo `main()` è una sola in Java
 - ▶ `public static void main(String [])` è l'unica signature valida
 - ▶ Mentre in C sia `int main(void)` che `int main(char **, int)` sono ugualmente accettabili
- Gli argomenti con cui un programma Java viene invocato vengono passati come parametri attraverso l'array (`String[] args`) che il metodo `main()` prende in ingresso
 - ▶ Nonostante sia un parametro del *metodo principale* di qualunque programma Java, si tratta di un comune array senza alcuna particolarità.



- 1 Stili e Convenzioni per il codice sorgente
- 2 Visual Studio Code (VSCode) e Debugging
- 3 Compilazione ed esecuzione avanzata in Java
- 4 Esecuzioni di programmi java con argomenti
- 5 Laboratorio**
 - Appendice: richiami utili per gli esercizi del Lab02

Preparazione ambiente di lavoro

- Accedere al PC di laboratorio con le proprie credenziali istituzionali
- Accedere al sito del corso
- Scaricare il materiale dell'esercitazione odierna
- Spostare il file scaricato sul Desktop
- Decomprimere il file e aprire con Visual Studio Code la directory ottenuta (File → Open Folder...)
- Puntare il terminale alla directory con i sorgenti dell'esercitazione odierna



Outline

- 1 Stili e Convenzioni per il codice sorgente
- 2 Visual Studio Code (VSCode) e Debugging
- 3 Compilazione ed esecuzione avanzata in Java
- 4 Esecuzioni di programmi java con argomenti
- 5 Laboratorio
 - Appendice: richiami utili per gli esercizi del Lab02



Formula per il calcolo della varianza

Sia n il numero di elementi dell'array ed x_i l'elemento all'indice i dell'array, e μ la media dei valori del suddetto array. La varianza σ^2 può essere calcolata come:

$$\sigma^2 = \frac{\sum_{i=0}^{n-1} (x_i - \mu)^2}{n}$$

