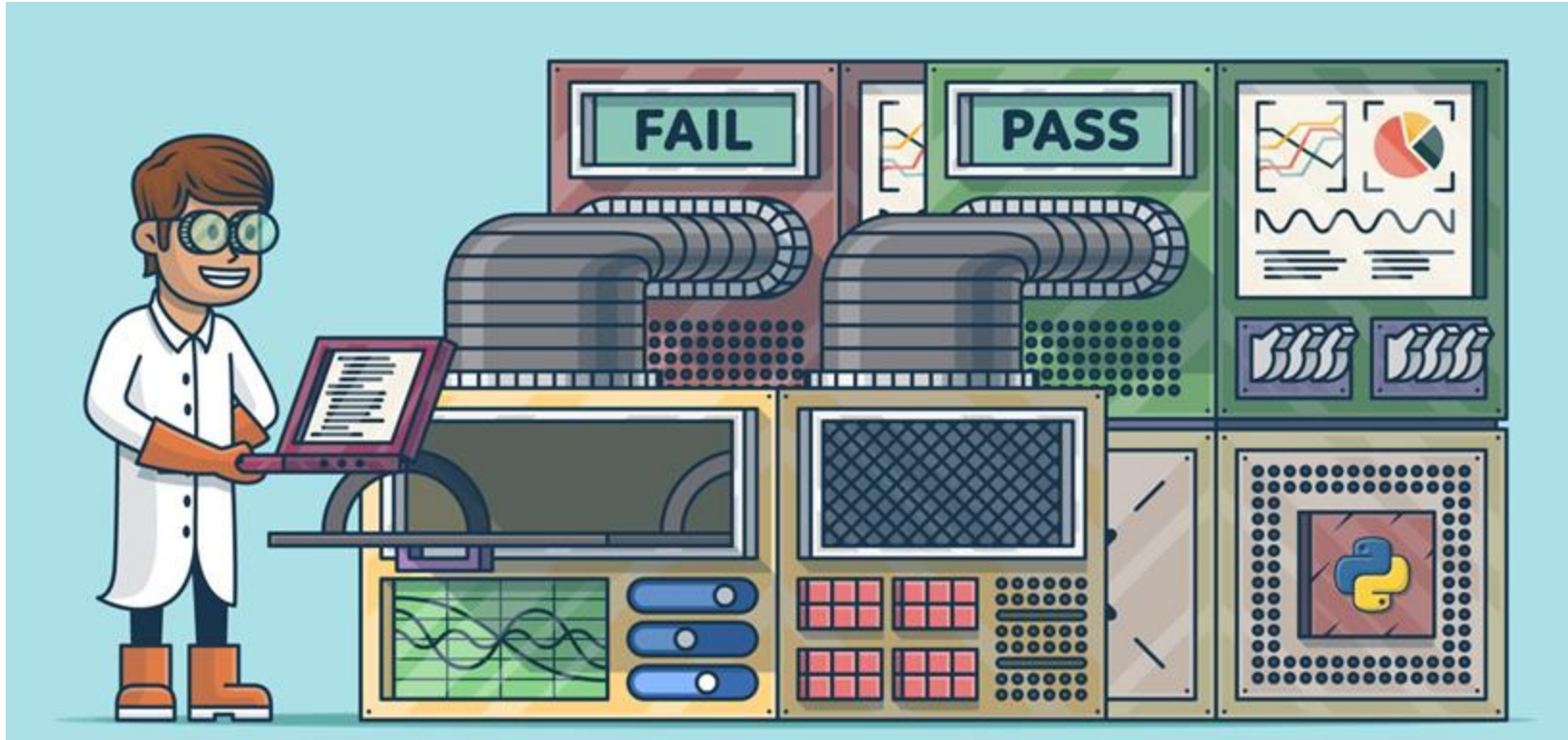


Testing

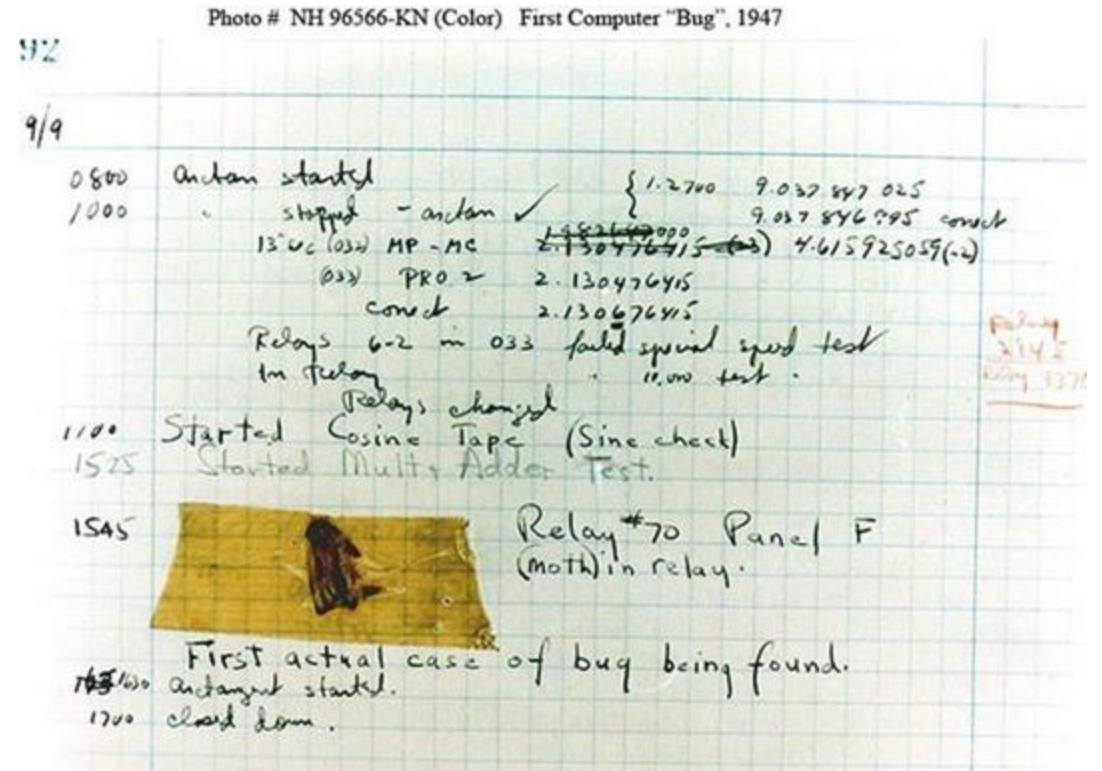


Agenda de la clase

- Que es hacer tests y que tipos de tests hay
- Tests de unidad
- Automatización de tests de unidad con Junit
- Elección y diseño de mis casos de test – dos estrategias
 - Particiones de equivalencia
 - Valores de borde
- Tests automatizados y cobertura

¿Qué es un bug/error?

- El programa no hace algo que debería hacer
- El programa hace algo mal
- El programa falla (revienta)



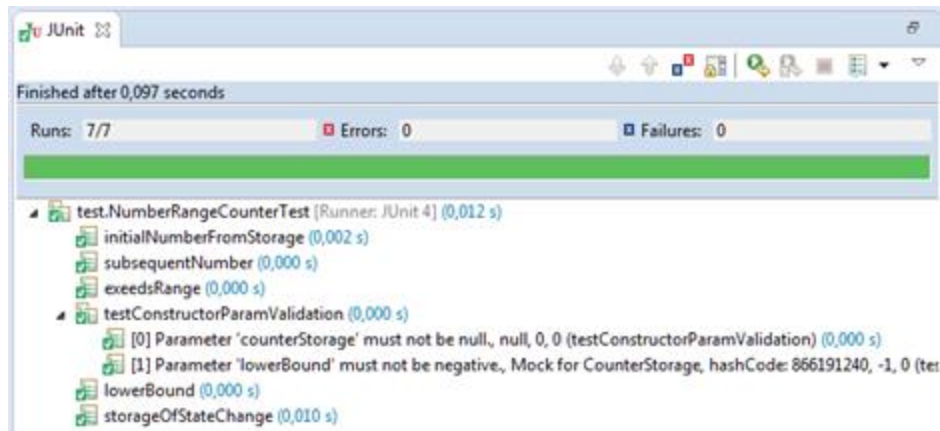
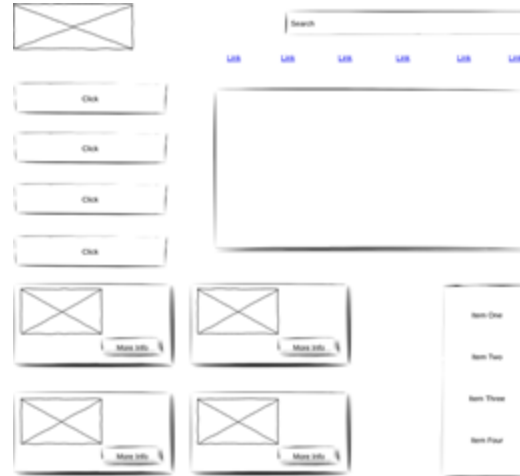
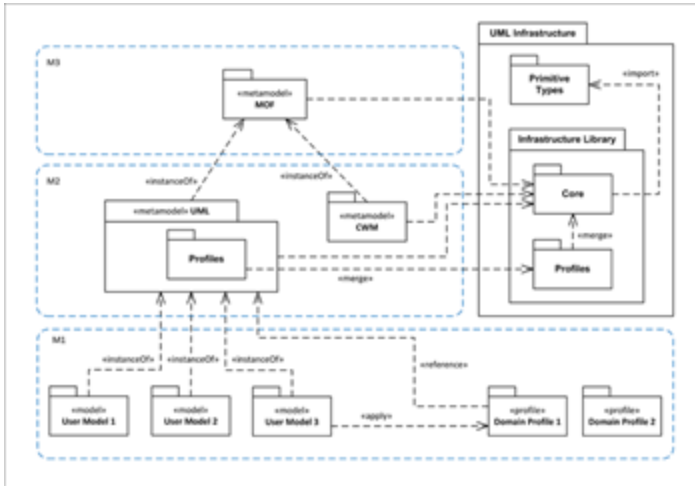
¿Qué es testear?

Asegurarse de que el programa:

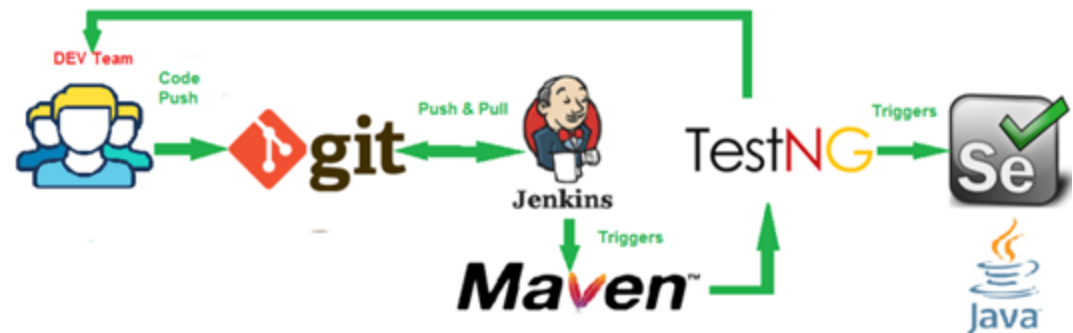
- hace lo que se espera
- lo hace como se espera y
- no falla



¿Para qué, con quien, cuándo, y como testear?



How Run Selenium Tests in Jenkins Using Maven



Tipos de test

- Tests funcionales
 - Test no funcionales
 - Tests de unidad
 - Tests de integración
 - Tests de regresión
 - Test punta a punta
 - Tests automatizados
- Test de carga
 - Test de performance
 - Test de aceptación
 - Test de UI
 - Test de accesibilidad
 - Alpha y beta tests
 - Test A/B
 - ...

¿Por qué no testeamos (o lo hacemos mal)?

- Lo dejamos para el final (¿para no trabajar de gusto?)
- Hay muchas combinaciones que considerar
- Requiere planificación, preparación y recursos adicionales
- Es una tarea repetitiva, y nos parece poco interesante
- Creemos que es tarea de otro, nosotros programamos (¿?)
- Creemos que alcanza con “programar bien”
- El objetivo de testear es encontrar bugs (¿será que eso nos molesta?)

Test de unidad

- Test que asegura que la unidad mínima de nuestro programa funciona correctamente, y aislada de otras unidades
 - En nuestro caso, la unidad de test es el método
- Testear un método es confirmar que el mismo acepta el rango esperado de entradas, y que retorna el valor esperado en cada caso
 - tengo en cuenta parámetros,
 - estado del objeto antes de ejecutar el método,
 - objeto que retorna el método, y
 - estado del objeto al concluir la ejecución del método

Tests automatizados

- Se utiliza software para guiar la ejecución de los tests y controlar los resultados
- Requiere que diseñemos, programemos y mantengamos programas “tests”
 - En nuestro caso, esos programas serán objetos
- Suele basarse en herramientas que resuelven gran parte del trabajo
- Una vez escritos, los puedo reproducir a costo mínimo, cuando quiera
- Los tests son “parte del software” (y un indicador de su calidad)

Automatizando tests de unidad



jUnit

- jUnit es un framework, en Java, para automatizar la ejecución de tests de unidad
- Ayuda a escribir tests útiles
- Cada test se ejecuta independientemente de otros (aislados)
- jUnit detecta, recolecta, y reporta errores y problemas
- xUnit es su nombre genérico; lo que aprendamos podemos llevarlo a otros lenguajes

Anatomía de un test suite junit

- Una clase de test por cada clase a testear
- Un método que prepara lo que necesitan los tests (el fixture)
 - Y queda en variables de instancia
- Uno o varios métodos de test por cada método a testear
- Un método que limpia lo que se preparó (si es necesario)

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

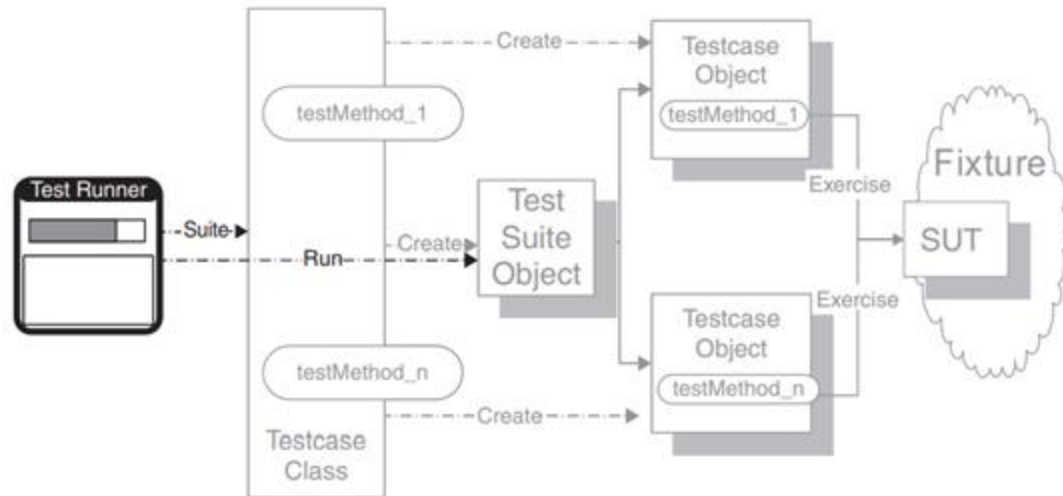
public class PersonaTest {

    Persona james;

    @BeforeEach
    void setUp() throws Exception {
        james = new Persona();
        james.setApellido("Glosing");
        james.setNombre("James");
    }

    @Test
    public void testNombreCompleto() {
        assertEquals("Glosing, James",
            james.getNombreCompleto());
    }
}
```

El test runner



```
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ ejemploTeoriaTesting ---
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running ar.edu.unlp.info.oo1.ejemploTeoriaTesting.RobotTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.026 s - in ar.edu.unlp.info.oo1.ejemploTeoriaTesting.RobotTest
[INFO] Results:
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.103 s
[INFO] Finished at: 2021-09-07T11:00:54-03:00
[INFO] -----
```

The screenshot shows an IDE window with the 'JUnit' tab selected. The 'Package Explorer' on the left shows the project structure. The main area displays the test results for 'RobotTest' [Runner: JUnit 5] (0.016 s). The results show two tests: 'testRetroceder()' (0.015 s) and 'testAvanzar()' (0.000 s), both passing. The status bar at the bottom indicates 'Finished after 0.082 seconds' and 'Runs: 2/2', 'Errors: 0', 'Failures: 0'.

Independencia entre tests

- No puedo asumir que otro test se ejecutó antes o se ejecutará después del que estoy escribiendo
- Por cada método de test (marcado con `@Test`):
 - Se crea una nueva instancia de nuestra clase de test
 - Se prepara (con el método marcado como `@BeforeEach`)
 - Se ejecuta el test y se registran errores y fallas

El Robot (ejemplo)



Robot
-position: int -energy: int
+getPosition(): int +getEnergy(): int +goForward() +goBackwards() -consumeEnergy()

- Nuestro robot avanza y retrocede de a un lugar
- En cada movimiento consume una unidad de energía
- ¿Qué tests deberíamos escribir?

Importamos las partes
de JUnit que necesitamos

```
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*;
```

Definición y preparación
del "fixture"

```
class RobotTest {  
  
    private Robot robot;  
  
    @BeforeEach  
    void setUp() {  
        robot = new Robot(0,100);  
    }
```

Ejercitar los objetos

Verificar resultados

```
    @Test  
    void testGoForward() {  
        robot.goForward();  
        assertEquals(1, robot.getPosition());  
    }
```

Ejercitar los objetos

Verificar resultados

```
    @Test  
    void testGoBackwards() {  
        robot.goBackwards();  
        assertEquals(-1, robot.getPosition());  
    }
```

Tests

Variantes del assert (algunas)

@Test

```
void assertExamples() {  
    assertEquals(5, "Hello".length());  
    assertNotEquals("Hello", "Bye");  
    assertNotNull(myList);  
    assertSame(myList, someList);  
    assertTrue(myList.isEmpty());  
    assertFalse(someList.isEmpty());  
    assertThrows(IndexOutOfBoundsException.class, () -> {  
        myList.remove("Hello");  
    });  
}
```

testingmain

Project

testing

src

main

java

robots 100% classes, 78% lines covered

Battery

InfiniteBattery 100% methods, 100% lines covered

SustainableRobot 66% methods, 66% lines covered

resources

test

java

robots

InfiniteBatteryTest

SustainableRobotTest

target

ry.java

SustainableRobot.java

SustainableRobotTest.java

5

6

7

8

9

10

11

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

Cobertura

- Cuanto testeamos nos interesa saber cuan completos/integrales son nuestros tests – podemos medirlo de distintas formas
 - Clases cubiertas, métodos cubiertos, líneas cubiertas
 - Condicionales (ver que se ejecutaron con true y falase)
 - Caminos/branches (ver si pasó por todos lados)
- Las herramientas modernas observan y reportan esos y otros valores
- Como escribir y mantener tests requiere esfuerzo no siempre maximizamos su cobertura cobertura
- Diseñar bien nuestros tests nos ayuda a enfocar el esfuerzo, optimizar el resultado y obtener un balance adecuado esfuerzo/cobertura

Pensando los tests



¿Por qué, cuándo, y como testear? (revisado)

- Testeamos para encontrar bugs
- Testeamos con un propósito (buscamos algo)
- Pensamos por qué testear algo y con qué nivel queremos hacerlo
- Testeamos temprano y frecuentemente
- Testeo tanto como sea el riesgo del artefacto
- No necesario testear código de base que otros ya testearon (por ejemplo, partes del SDK, etc.)

Estrategia general

- Pensar que podría variar (que valores puede tomar) y que pueda causar un error o falla
- Elegir valores (de estados y parámetros) de prueba para maximizar las chances de encontrar errores haciendo la menor cantidad de pruebas posibles
 - Una combinación de valores es “un caso de prueba”
- Nos vamos a enfocar en dos estrategias:
 - Particiones equivalentes
 - Valores de borde

Tests de particiones equivalentes

- Partición de equivalencia: conjunto de casos que prueban lo mismo o revelan el mismo bug
 - Asumo que si un ejemplo de una partición pasa el test, los otros también lo harán. Elijo uno.
- Si se trata de casos en un conjunto, tomo un caso que pertenezca al conjunto y uno que no
 - Ej., debe tener entrada -> Casos: una persona con entrada, una sin
- Si se trata de valores en un rango, tomo un caso dentro y uno por fuera en cada lado del rango
 - Ej., la temperatura debe estar entre 0 y 100 - > casos: -50, 50 , 150.
 - Veremos que estos casos pueden mejorarse

El Robot minimalista



MinimalRobot
-position: int
+getPosition(): int +goForward() +goBackwards()

- Nuestro robot avanza y retrocede de a un lugar sin importarle nada
- ¿Qué tests deberíamos escribir? (Qué clases y qué métodos)
- ¿Qué particiones identificamos?
- ¿Cuáles serían buenos casos de test para cada una?
- ¿Podemos pensar otros casos que prueben algo diferente?

MinimalRobot
-position: int
+getPosition(): int +goForward() +goBackwards()

- Identificamos dos métodos a testear
- Identificamos una sola partición
- Cualquier robot da lo mismo
- ¿Qué métodos cubrimos?

```
class MinimalRobotTest {  
  
    private MinimalRobot robot;  
  
    @BeforeEach  
    void setUp() {  
        robot = new MinimalRobot();  
    }  
  
    @Test  
    void goForward() {  
        robot.goForward();  
        assertEquals(1, robot.getPosition());  
    }  
  
    @Test  
    void goBackwards() {  
        robot.goBackwards();  
        assertEquals(-1, robot.getPosition());  
    }  
}
```

El Robot apagable



ToggleableRobot

-position: int
-isOn: boolean

+getPosition(): int
+toggle()
+goForward()
+goBackwards()

- Se puede encender y apagar (con un mismo mensaje)
- Si esta apagado, no hace nada al pedirle que se mueva
- ¿Qué tests deberíamos escribir?
- ¿Qué particiones identificamos?
- ¿Cuáles serían buenos casos de test para cada una?
- ¿Podemos pensar otros casos que prueben algo diferente?

ToggleableRobot
-position: int -isOn: boolean
+getPosition(): int +toggle() +goForward() +goBackwards()

- Identificamos dos métodos a testear
 - ¿Necesitamos testear toggle()?
- Identificamos dos particiones
 - Encendido, en cualquier lugar
 - Apagado, en cualquier lugar
- Cuatro casos en total
 - Podemos tener varios casos en un método de test
 - Podemos separar casos en métodos

```
class ToggleableRobotTest {  
    private ToggleableRobot onRobot, offRobot;  
  
    @BeforeEach  
    void setUp() {  
        onRobot = new ToggleableRobot();  
        onRobot.toggle();  
        offRobot = new ToggleableRobot();  
    }  
  
    @Test  
    void testGoForward() {  
        //Partition of robots that are turned on  
        onRobot.goForward();  
        assertEquals(1, onRobot.getPosition());  
        //Partition of robots that are turned off  
        offRobot.goForward();  
        assertEquals(0, offRobot.getPosition());  
    }  
  
    @Test  
    void testGoBackwards_onRobots() {  
        onRobot.goBackwards();  
        assertEquals(-1, onRobot.getPosition());  
    }  
}
```

Tests con valores de borde

- Los errores ocurren con frecuencia en los límites y ahí es donde los vamos a buscar
- Intentamos identificar bordes en nuestras particiones de equivalencia y elegimos esos valores
- Buscar los bordes en estados/parámetros: velocidad, cantidad, posición, tamaño , duración, edad, etc.
 - También podemos buscar en relaciones entre ellas (diferencia entre saldo y monto a extraer)
- Y buscar valores como: primero/último, máximo/mínimo, arriba/abajo, principio/fin, vacío/lleño, antes/después, junto a, alejado de , etc.

El Robot positivista



PositivistRobot

-position: int

+getPosition(): int

+goForward()

+goBackwards()

- Nuestro robot avanza y retrocede de a un lugar, **pero solo en positivos (desde 0 a Integer.MAX_VALUE)**
- ¿Qué particiones identificamos?
- ¿Cuáles son los bordes?
- ¿Podemos pensar otros casos que prueben algo diferente?

PositivistRobot
-position: int
+getPosition(): int +goForward() +goBackwards()

- Identificamos dos métodos a testear
- ¿Qué particiones/bordes encontramos para cada método?
 - Considero estado, parámetros y semántica del método
- Las particiones/bordes pueden ser diferentes para distintos métodos

```

class PositivistRobotTest {
    PositivistRobot robotAtZero, robotAtMax, robotInBetween;
    @BeforeEach
    void setUp() {
        robotAtZero = new PositivistRobot();
        robotAtMax = new PositivistRobot(Integer.MAX_VALUE);
        robotInBetween = new PositivistRobot(100);
    }

    @Test
    void testGoBackwards_inBetween() {
        robotInBetween.goBackwards();
        assertEquals(99, robotInBetween.getPosition());
    }

    @Test
    void testGoBackwards_atZero() {
        robotAtZero.goBackwards();
        assertEquals(0, robotAtZero.getPosition());
    }

    @Test
    void testGoForward_inBetween() {...}
    @Test
    void testGoForward_atMax() {...}

```

El saltarín y hambriento



JumpingHungryRobot
-position: int -hunger: int -energy: int
+getPosition(): int +jumpForward(places: int) +jumpBackwards(places: int) +charge(amount: int): int +setHunger(hunger: int)

- Tiene energía (que puede ser 0), y tiene hambre (mínimo 1)
- Cada lugar que se mueve usa tanta energía como su hambre indica
- Si no tiene energía suficiente se queda en el lugar (aunque le pida que avance o retroceda)
- En lugar de avanzar y retroceder de a uno, salta

JumpingHungryRobot
-position: int -hunger: int -energy: int
+getPosition(): int +jumpForward(places: int) +jumpBackwards(places: int) +charge(amount: int): int +setHunger(hunger: int)

- Identificamos tres métodos a testear: charge y los dos jump
- ¿Qué particiones/bordes encontramos para cada método?
 - Pienso en combinaciones de carga, hambre y cantidad de lugares

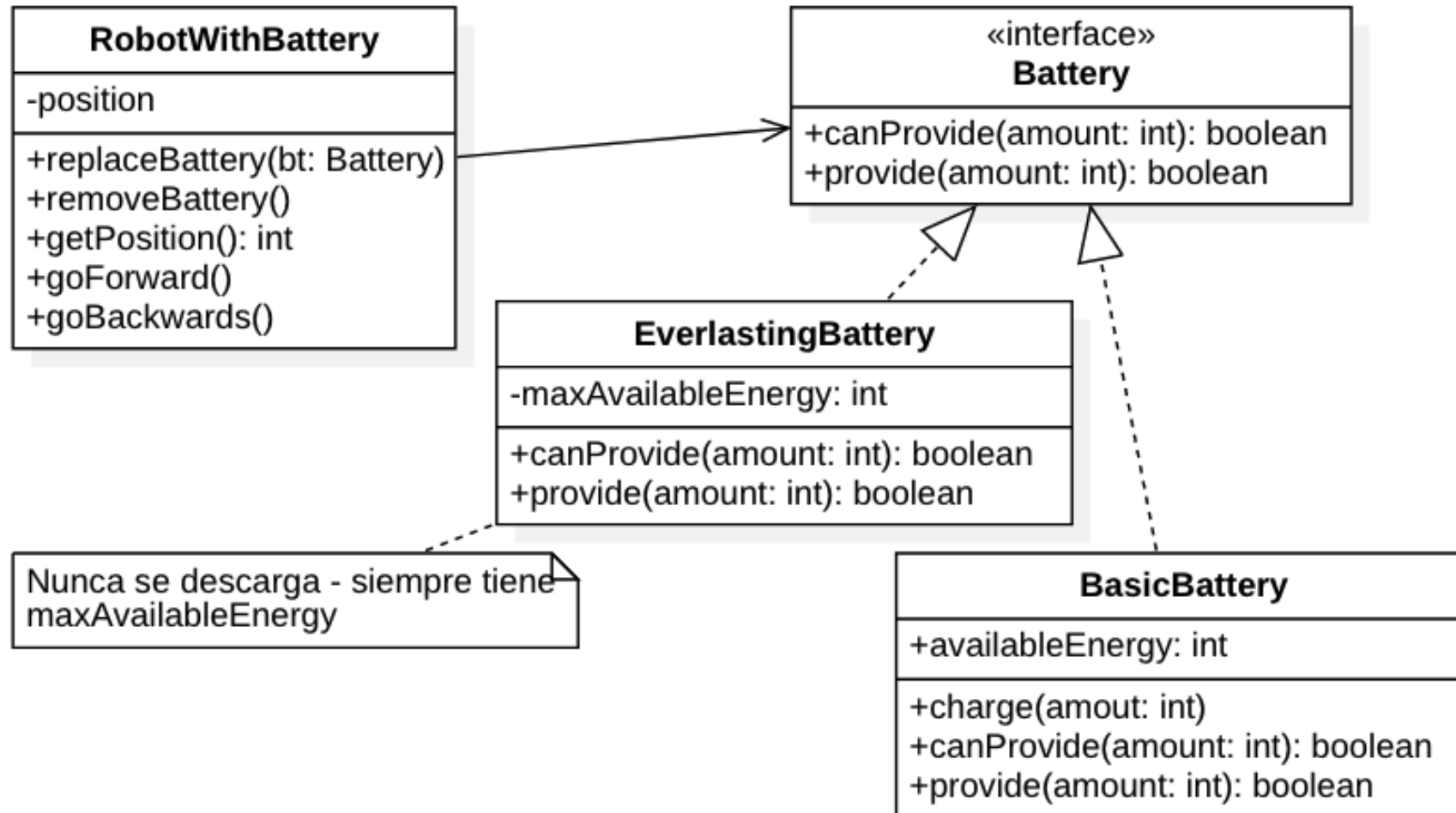
- testCharge()
 - position y hunger son irrelevantes
 - Cualquier combinación energía/amount sirve (¿no?)
 - Elijo: energy = 0; amount = 1;
- testJump...()
 - Considero la relación energy, hunger, y places
 - Partición sin suficiente energía
 - Elijo uno de los casos mínimos
 - energy = 0; hunger = 1; places = 1
 - Partición con suficiente energía
 - Elijo uno de los casos mínimos
 - energy = 1; hunger = 1; places = 1
 - ¿algo más?

Cuanto se les pida
“identifique, especifique y
justifique los casos de test”,
se espera que respondan
algo como esto ...

- Identificamos tres métodos a testear: charge y los dos jump
- ¿Qué particiones/bordes encontramos para cada método?
 - Pienso en combinaciones de carga, hambre y cantidad de lugares

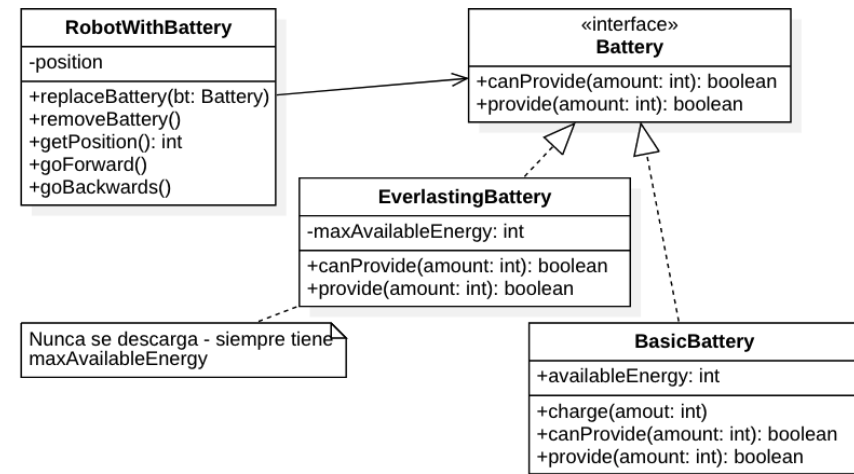
- testCharge()
 - position y hunger son irrelevantes
 - Cualquier combinación energía/amount sirve (¿no?)
 - Elijo: energy = 0; amount = 1;
- testJump...()
 - Considero la relación energy, hunger, y places
 - Partición sin suficiente energía
 - Elijo uno de los casos mínimos
 - energy = 0; hunger = 1; places = 1
 - Partición con suficiente energía
 - Elijo uno de los casos mínimos
 - energy = 1; hunger = 1; places = 1
 - ¿algo más?

¿Qué testeamos en este caso y cómo?

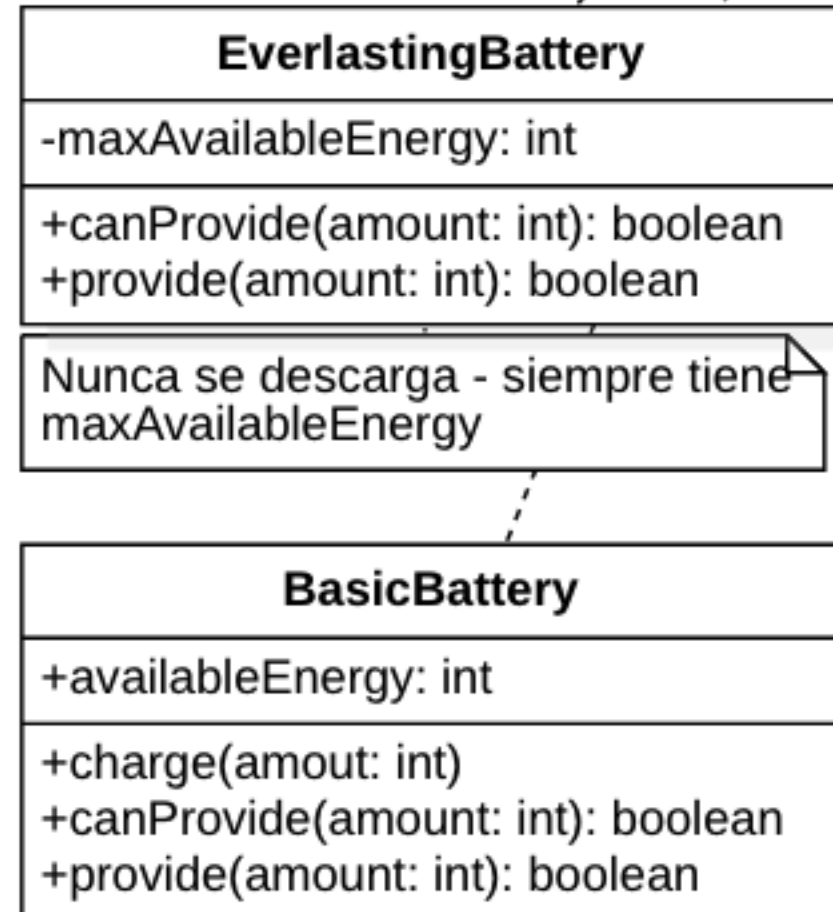


¿Qué testeamos en este caso y cómo?

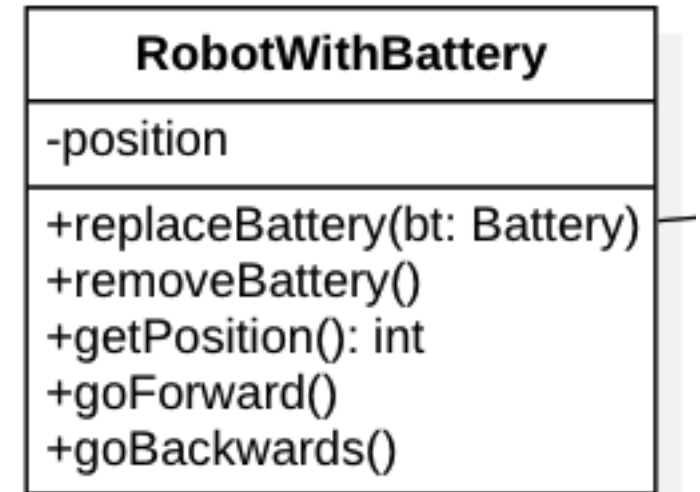
- Testeamos cada clase por separado
- Prestamos atención a los casos que son relevantes en cada clase
- Atención: ¡ al testear RobotWithBattery, evitamos redundar en tests de las baterías ! Eso ya esta testeado.



- EverlastingBattery
 - canProvide() y provide() con:
 - amount = 1 & amount = 2
 - maxAvailableEnergy = 1 ;
- BasicBattery
 - canProvide() y provide() con:
 - amount = 1;
 - availableEnergy = 0 & 1 ;



- RobotWithBattery
 - goForward() y goBackwards()
- Pensando en la relación energía-consumo
 - En cualquier lugar - elijo position = 0
 - Sin suficiente energia (cualquier bateria)
 - Elijo una BasicBaterly con energy = 0
 - Con suficiente energia (cualquier bateria)
 - Elijo una BasicBaterly con energy = 1 (un límite)
- En los límites & suficiente energia
 - Elijo una BasicBaterly con energy = 1 (un límite)
 - goForward() en Integer.MAX_VALUE
 - goBaclwards() en Integer.MAX_VALUE



Testing en OO1

- En el marco de OO1, testear es asegurarnos de que nuestros objetos hacen lo que se espera, como se espera
- Escribir tests de unidad (con JUnit) es parte “programar”
- Escribir tests nos ayuda a entender que se espera de nuestros objetos
- Con lo que sabemos hasta ahora encontraremos situaciones complejas de resolver
 - Ya veremos en OO2 estrategias para atacarlas
 - Por ahora el foco es testear con propósito, y diseñar bien los tests/casos

Bibliografía

