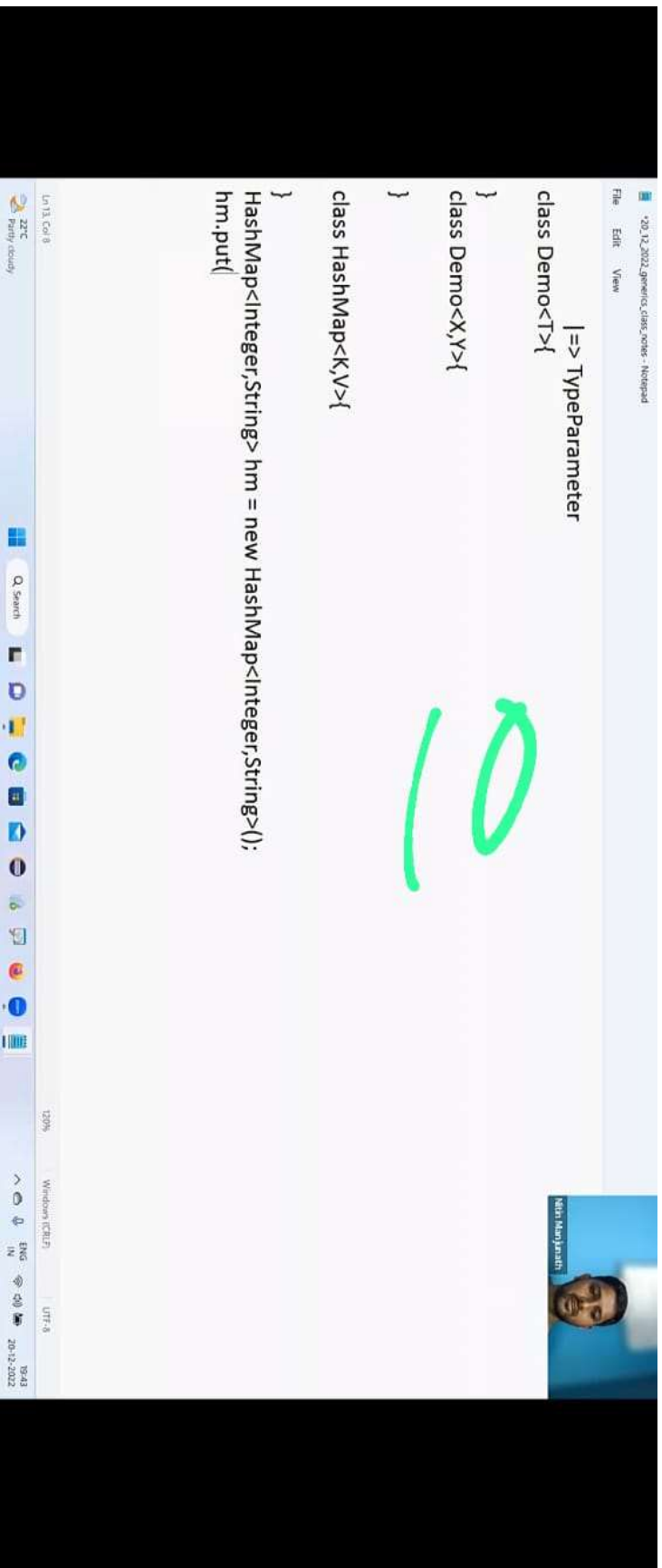


## Jenerics Part 2



```
*20_12_2022_generics.class notes - Notepad
File Edit View

I=> TypeParameter
class Demo<T>{
}
class Demo<X,Y>{
}
class HashMap<K,V>{
}
HashMap<Integer,String> hm = new HashMap<Integer,String>();
hm.put(
```

Ln 13, Col 8

100% Windows (CTRL) | UTF-8

19:43 20-12-2022

Notepad

10

Video Feed: Notepad

keyPoints about BoundedTypes

=> As the type parameter we can use any valid java identifier but it convention to use T always.

```
eg: class Test<T>{}
```

```
class Test<INeuron>{}
```

=> We can pass any no of type parameters need not be one.

```
eg: class HashMap<K,V>{}
```

```
HashMap<Integer,String> h=new HashMap<Integer,String>();
```



Which of the following are valid?

class Test <T extends Number&Runnable> {} //valid

Number -> class

Runnable -> interface

class Test<T extends Number&Runnable&Comparable> {} //valid

Number -> class

Runnable -> interface

Comparable -> interface

class Test<T extends Number&String> {} //invalid

we can't extend more than one class at a time.

class Test<T extends Runnable&Comparable> {} | 1

class Test<T extends Runnable&Number> {}

2



### Runnable-> interface

class Test<T extends Number&Runnable&Comparable> {} //valid

Number -> class

Runnable-> interface

Comparable -> interface

class Test<T extends Number&String> {} //invalid  
we can't extend more than one class at a time.

class Test<T extends Runnable&Comparable> {} //valid

Runnable-> interface

Comparable -> interface

class Test<T extends Runnable&Number> {} //invalid

Runnable-> interface

Number -> class

rule: first inherit and the implement so invalid



class Test<T extends Runnable&Comparable> {} //valid

Runnable-> interface

Comparable -> interface

class Test<T extends Runnable&Number> {} //invalid

Runnable-> interface

Number -> class

rule: first inherit and the implement so invalid

GenericClass

=====

class: Type parameter

Can we apply TypeParameter at MethodLevel?

Ans. Yes, it is possible.

Handwritten green text: "Yes"



0.027 MW  
Unitless - Power

0.00017 W  
Unitless - Power

File View

Clipboard

Image

Tools

Brushes

Shapes

Size

Colors

Console

16

B I U S

Background fill

ArrayList<String> a1 = new ArrayList<String>();  
a1.add("sachin");  
a1.add("dhoni");  
};  
  
m1(a1);  
  
ArrayList<Integer> a11 = new ArrayList<Integer>();  
a11.add(10);  
a11.add(20);  
};  
  
m2(a11);

void m1(ArrayList<String> a1){  
};  
}  
  
void m2(ArrayList<Integer> a1){  
};  
}

15

100%

19:56

20-12-2022

1. methodOne(ArrayList<String>a1):

This method is applicable for ArrayList of only String type.

```
methodOne(ArrayList<String> a1){
```

```
    al.add("sachin");
```

```
    al.add("navinreddy");
```

```
    al.add("iNeuron");
```

```
    al.add(new Integer(10)); //invalid
```

```
}
```

2. methodOne(ArrayList<?>a1):

This method is applicable for ArrayList of any type.

```
methodOne(ArrayList<String> a1){
```

```
    al.add("sachin");
```

```
    al.add("navinreddy");
```

```
    al.add("iNeuron");
```

```
    al.add(new Integer(10)); //invalid
```

```
}
```



```
al.add("navinreddy");  
al.add("iNeuron");  
al.add(new Integer(10)); // invalid  
}
```

## 2. methodOne(ArrayList<?> al):

This method is applicable for ArrayList of any type.

```
methodOne(ArrayList<?> al){  
    al.add("navinreddy"); // invalid  
    al.add("iNeuron"); // invalid  
    al.add(new Integer(10)); // invalid  
}
```





```
al.add("navinreddy");  
al.add("iNeuron");  
al.add(new Integer(10)); // invalid  
}
```

2. methodOne(ArrayList<?>a1):

This method is applicable for ArrayList of any type.

This method signature is best suited only when we perform read operation on ArrayList.

```
methodOne(ArrayList<?>a1){  
    al.add("navinreddy");//invalid  
    al.add("iNeuron");//invalid  
    al.add(new Integer(10)); //invalid  
    al.add(null);  
}
```

81



```
20_12_2022, genetics class notes - Notepad
File Edit View
al.add("navinreddy");
al.add("iNeuron");
al.add(new Integer(10)); // invalid
}
```

Within the method we can add only String type of objects and null to the List.

methodOne(ArrayList<?> l):

We can use this method for ArrayList of any type but within the method we can't add anything to the List except null.

Example:

```
l.add(null); // (valid)
l.add("A"); // (invalid)
l.add(10); // (invalid)
```

This method is useful whenever we are performing only read operation.

19



### 3. methodOne(ArrayList<? extends X> a)

X -> class, we can make a call to method by passing ArrayList of X type or its Child type.

X -> interface, we can make a call to method by passing ArrayList of X type or its Implementation class.

```
methodOne(ArrayList<? extends X> a){
```

```
    al.add(null);
```

```
}
```

Best suited only for read operation.

### 4. methodOne(ArrayList<? super X> al)

X -> class, we can make a call to method by passing ArrayList of X type or its super class

X -> interface, we can make a call to method by passing ArrayList of X type or its super class of implementation class of X.

```
methodOne(ArrayList<? super X> al){
```

```
    al.add(X);
```

```
    al.add(null);
```

```
}
```

21



Note:

Which of the following declarations are allowed?

1. `ArrayList<String> l1=new ArrayList<String>();` //valid
2. `ArrayList<?> l2=new ArrayList<String>();` //valid
3. `ArrayList<?> l3=new ArrayList<Integer>();` //valid
4. `ArrayList<? extends Number> l4=new ArrayList<Integer>();` //valid
5. `ArrayList<? extends Number> l5=new ArrayList<String>();` //invalid
6. `ArrayList<?> l6=new ArrayList<? extends Number>();` //invalid
7. `ArrayList<?> l7=new ArrayList<?>();` //invalid

22



```
class Demo<T>{
```

|=> Type parameter defined just before the return type

```
public <T> void m1(T t){
```

```
}
```

```
}
```

Which of the following declarations are allowed?

```
public<T> void methodOne1(T t){}
```

I

```
public<T extends Number> void methodOne2(T t){}
```

```
public<T extends Number&Comparable> void methodOne3(T t){}
```

```
public<T extends Number&Comparable&Runnable> void methodOne4(T t){}
```

```
public<T extends Number&Thread> void methodOne(T t){}
```

```
public<T extends Runnable&Number> void methodOne(T t){}
```

```
public<T extends Number&Runnable> void methodOne(T t){}
```

24



20-12-2022 generic\_array.jpg - Paint

FileView

Clipboard

Image

Tools

Brushes

Shapes

Size

Colors

Console

14

B

I

U

S

=

=

Background fill

ArrayList<String> a1 = new ArrayList<String>();  
a1.add("sachin");  
a1.add("dhoni");  
;;;;  
ml(a1);  
ArrayList<Integer> a11 = new ArrayList<Integer>();  
a11.add(10);  
a11.add(20);  
;;;;  
ml(a11);  
1. ml(ArrayList<String> a1)  
2. ml(ArrayList<> a1)  
3. ml(ArrayList<? extends X> a1)  
4. ml(ArrayList<? super X> a1)

GenericMethod with wild card pattern

void ml(ArrayList<? > a1){  
    ~~String~~ Integer  
    a1.add("sachin");  
    a1.add("navinreddy");  
    a1.add("hyder");  
    a1.add(new Integer(10)); //invalid  
}

void ml(ArrayList<? a1){  
    while(itr.hasNext()){  
        Object data = itr.next();  
        System.out.println(data);  
    }  
}

20

100%

ENG

IN

20-12-2022

File Edit View Search Document Project Tools Browser Export Window Help

Directory Object Functions

D:\New Volume

Wrappers\classes

ItemTest.java

Test.java

ItemTest.java

Test.java

5 //MethodOverloading

6 public void m1(ArrayList a){}

7 public void m1(ArrayList b){}

8

9 public static void main(String[] args)

10 {

11

12

13 }

14

15 }

16 // 1. Compiler will scan the code

17 // 2. Check the argument type

18 // 3. if Generics found in the argument type remove the Generics syntax

19 // 4. Compiler will again check the syntax

20

21

22 Arrays

23 Collection

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

14



```
class Demo<T>{
```

|=> Type parameter defined just before the return type

```
public <T> void m1(T t){
```

```
}
```

```
}
```

Which of the following declarations are allowed?

```
public <T> void methodOne1(T t){} //valid
```

```
public<T extends Number> void methodOne2(T t){} //valid
```

```
public<T extends Number&Comparable> void methodOne3(T t){} //valid
```

```
public<T extends Number&Comparable&Runnable> void methodOne4(T t){} //valid
```

```
public<T extends Number&Thread> void methodOne(T t){} //invalid
```

```
public<T extends Runnable&Number> void methodOne(T t){} //invalid
```

```
public<T extends Number&Runnable> void methodOne(T t){} //valid
```

25





Code editor window showing Java code for a test class:

```
1 import java.util.*;
2
3 class Test
4 {
5     public static void main(String[] args)
6     {
7         ArrayList<? super Number> al = new ArrayList<Object>();
8     }
9 }
10
11
```

A large green handwritten number "26" is visible in the center of the code editor.

Below the code editor, there are three Notepad windows open:

- generic imports flow - Notepad
- Generic - Notepad
- 20.12.2022, generic.cla... - X

The taskbar at the bottom shows the system clock as 20:41 on 20-12-2022, and the language is set to ENG.

```

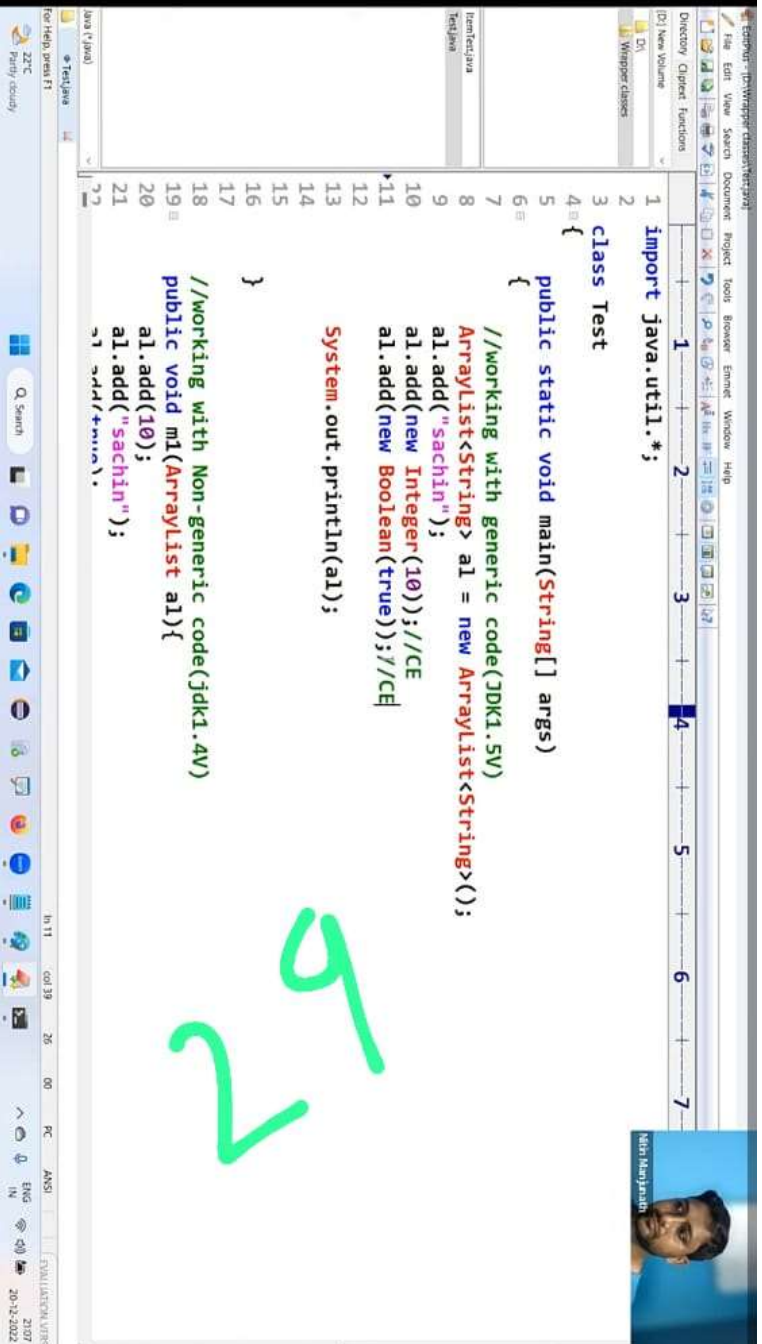
public int indexOf(java.lang.Object);
public int lastIndexOf(java.lang.Object);
public java.lang.Object clone();
public java.lang.Object[] toArray();
public <T> T[] toArray(T[]);
E elementData(int);
public E get(int);
public E set(int, E);
public boolean add(E);
public void add(int, E);
public E remove(int);
public boolean remove(java.lang.Object);
public void clear();
public boolean addAll(java.util.Collection<? extends E>);
public boolean addAll(int, java.util.Collection<? extends E>);
protected void removeRange(int, int);
public boolean removeAll(java.util.Collection<?>);
public boolean retainAll(java.util.Collection<?>);
public java.util.ListIterator<E> listIterator(int);
public java.util.ListIterator<E> listIterator();
public java.util.Iterator<E> iterator();
public java.util.List<E> subList(int, int);
static void forEach(java.util.function.Consumer<? super E>);
public java.util.Spliterator<E> spliterator();
public boolean removeIf(java.util.function.Predicate<? super E>);
public void replaceAll(java.util.function.UnaryOperator<E>);
public void sort(java.util.Comparator<? super E>);
static int access$000(java.util.ArrayList);
}

```

27



Nitin Rajgurun



## TypeParameter at Method level

=====

|=> TypeParameter

class Demo<T>{

|=> Type parameter defined just before the return type

public <T> void m1(T t){

}

25



Editor - [D:\Wrapper classes\Test.java]

File Edit View Search Document Project Tools Browser Emmet Window Help

Directory Content Functions

D:\New Volume

Wrapper classes

Test.java

1 2 3 4 5 6 7

```
4 = {
5
6 = {
7
8
9
10
11
12
13
14
15
16
17
18
19 =
20
21
22
23
24
```

```
public static void main(String[] args)
{
    //working with generic code(JDK1.5V)
    ArrayList<String> a1 = new ArrayList<String>();
    a1.add("sachin");

    m1(a1);

    System.out.println(a1);//["sachin",10,"dhoni",true]

}

//working with Non-generic code(JDK 1.4V)
public static void m1(ArrayList a1){//ArrayList a1 = new ArrayList<String>();
    a1.add(10);
    a1.add("dhoni");
    a1.add(true);
}
```

Test.java

Test.java

22°C

20-12-2022

15

cod 9

26

00

PC

ANSI

ENG

IN

20-12-2022



OT



```
D:\Wrapper\classes>javac Test.java
Note: Test.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\Wrapper\classes>java Test
[sachin, 10, dhoni, true]

D:\Wrapper\classes>
```

15

Editor - (D:\Wrapper classes\Test.java 1)

File Edit View Search Document Project Tools Browser Export Window Help

Directory Content Functions

D:\ New Volume

Wrappers\classes

Item Test.java

Test.java

```
1 import java.util.*;
2
3 class Test
4 {
5     public static void main(String[] args)
6     {
7         ArrayList a1 = new ArrayList<Integer>();
8         a1.add("sachin");
9         a1.add("dhoni");
10
11         ArrayList a12 = new ArrayList<String>();
12         a12.add(10);
13         a12.add(10.5);
14         a12.add("sachin");
15
16         //Generics concept is applicable at Compiler level and at the runtime
17         // We don't have the concept of generics
18         ArrayList a1 = new ArrayList();
19
20
21
22
```

2

Man Harjanto

17 27 00 PC ANSI EVALUATING...

27°C 21:22 20-12-2022





```
}  
    l.add(true);  
}
```

#### Conclusions :

Generics concept is applicable only at compile time, at runtime there is no such type of concept.

At the time of compilation, as the last step generics concept is removed, hence for jvm generics syntax won't be available.

Hence the following declarations are equal.

```
ArrayList l=new ArrayList<String>();  
ArrayList l=new ArrayList<Integer>();  
ArrayList l=new ArrayList<Double>();
```

1

All are equal at runtime, becoz compiler will remove these generics syntax

ArrayList l=new ArrayList();

Example 1:  
import java.util.\*;

```
class Test {  
    public static void main(String[] args) {
```



### CE: duplicate methods found

Behind the scenes by the compiler

=====

1. Compiler will scan the code
2. Check the argument type
3. if Generics found in the argument type remove the Generics syntax
4. Compiler will again check the syntax

### Example3:

The following 2 declarations are equal.

```
ArrayList<String> l1=new ArrayList();
```

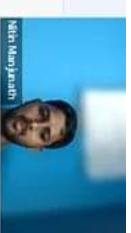
```
ArrayList<String> l2=new ArrayList<String>();
```

For these ArrayList objects we can add only String type of objects.

```
l1.add("A");//valid
```

```
l1.add(10);//invalid
```

→



```
D:\Wrapper\classes>javap java.util.TreeSet
Compiled from "TreeSet.java"
public class java.util.TreeSet<E> extends java.util.AbstractSet<E> implements java.util.NavigableSet<E>, java.lang.NavigableSet<E>, java.lang.NavigableMap<E, java.lang.Object>
    Serializable {
    java.util.TreeSet(java.util.NavigableMap<E, java.lang.Object>);
    public java.util.TreeSet();
    public java.util.TreeSet(java.util.Comparator<? super E>);
    public java.util.TreeSet(java.util.Collection<? extends E>);
    public java.util.TreeSet(java.util.SortedSet<E>);
    public java.util.Iterator<E> iterator();
    public java.util.Iterator<E> descendingIterator();
    public java.util.NavigableSet<E> descendingSet();
    public int size();
    public boolean isEmpty();
    public boolean contains(java.lang.Object);
    public boolean add(E);
    public boolean remove(java.lang.Object);
    public void clear();
    public boolean addAll(java.util.Collection<? extends E>);
    public java.util.NavigableSet<E> subset(E, boolean);
    public java.util.NavigableSet<E> headSet(E, boolean);
    public java.util.NavigableSet<E> tailSet(E, boolean);
    public java.util.SortedSet<E> subset(E, E);
    public java.util.SortedSet<E> headSet(E);
    public java.util.SortedSet<E> tailSet(E);
    public java.util.Comparator<? super E> comparator();
    public E first();
    public E last();
    public E lower(E);
    public E floor(E);
    public E ceiling(E);
    public E higher(E);
```

85



```
l1.add("A");//valid  
l1.add(10);//invalid
```

### Comparable vs Comparator

=====

public TreeSet();

|=> When we use the above constructor, JVM will internally use Comparable interface method to sort the Objects based on default natural sorting order.

What is Comparable interface? |

It is a functional interface present in java.lang package.

This interface is internally used by TreeSet object during sorting process.

@FunctionalInterface

```
public interface java.lang.Comparable<T> {
```

```
    public abstract int compareTo(T);
```

```
}
```

59





```
ts.add("Z");  
ts.add("L");  
ts.add("B");  
ts.add(null); //NullPointerException  
ts.add(10); //ClassCastException  
  
System.out.println(ts); // [A, B, L, Z]  
  
}
```

Note:

If we are keeping the data inside TreeSet object, then the data should be

- Homogenous ==> because it uses compareTo() to sort the Object
- The object should compulsorily implements an interface called "Comparable".  
if we fail to do so, it would result in "ClassCastException".

7



 Command Prompt

Note: `test.java` uses unchecked or unsafe operations.  
Note: Recompile with `-Xlint:unchecked` for details.

```
D:\Wrapper classes>java Test
```

```
Exception in thread "main" java.lang.ClassCastException: java.lang.StringBuffer cannot be cast to java.lang.Comparable
    at java.util.TreeMap.compare(TreeMap.java:1294)
    at java.util.TreeMap.put(TreeMap.java:588)
    at java.util.TreeSet.add(TreeSet.java:285)
    at Test.main(Test.java:18)
```

```
D:\Wrapper\classes>javap java.lang.String
```

```
Compiled from "String.java"
public final class java.lang.String implements java.io.Serializable, java.lang.Comparable<java.lang.String>, java.lang.CharSequence {
    public static final java.lang.String CASE_INSENSITIVE_ORDER;
```

```

public java.lang.String(java.lang.String);
public java.lang.String(char[]): int, int);
public java.lang.String(int): int, int);
public java.lang.String(byte[]): int, int);
public java.lang.String(byte[], int, int, java.lang.String) throws java.io.UnsupportedEncodingException;
public java.lang.String(byte[], java.lang.String) throws java.io.UnsupportedEncodingException;
public java.lang.String(byte[], java.nio.charset.Charset): int, int);
public java.lang.String(byte[]): int, int);
public java.lang.String(byte[]): int, int);
public java.lang.String(java.lang.StringBuffer);
public java.lang.String(java.lang.StringBuilder);
public java.lang.String(char[], boolean);
public int length();
public boolean isEmpty();
public char charAt(int);
public int codePointAt(int);
public int codePointBefore(int);
public int codePointCount(int, int);
public int offsetByCodePoints(int, int);
void getChars(int, int, char[], int);
public void getChars(int, int, char[], int);

```

Encapsulation:





```
ts.add(new StringBuffer("A"));  
ts.add(new StringBuffer("Z"));  
ts.add(new StringBuffer("L"));  
ts.add(new StringBuffer("B"));
```

```
System.out.println(ts);//ClassCastException
```

```
}
```

```
}
```

note: All Wrapper classes and String class has implemented "Comparable" interface.

StringBuffer class has not implemented Comparable interface, so the above program would result in "ClassCastException".





Question 15:

What will be the result of compiling and executing Test class?

```
public class Test {  
    public static void main(String[] args) {  
        try {  
            main(args); //infinite-call to the same main()====> StackOverflowError(Child of Error)  
        } catch (Exception ex) {  
            System.out.println("CATCH-");  
        }  
        System.out.println("OUT");  
    }  
}
```

- A. CATCH-OUT
- B. OUT
- C. None of the System.out.println() will be executed
- D. CompilationError. I

Answer: C

44



```
public class Test {  
    public static void main(String[] args) {  
        StringBuilder sb = new StringBuilder(); // capacity = 16  
        try {  
            for(;;) {  
                sb.append("Neuron"); // memory in heap area in finite (OutOfMemoryError) ==> Error class  
            }  
        } catch (Exception e) {  
            System.out.println("Exception!!!");  
        }  
        System.out.println("Main ends!!!");  
    }  
}
```

AS

What will be the result of compiling and executing Test class?

- A. "Main ends!!!" printed on the console and program terminates successfully.
- B. "Exception!!!" and "Main ends!!!" printed on the console and program terminates successfully.
- C. "Exception!!!" is printed on the console and program terminates successfully.
- D. "Exception!!!" is printed on the console and program terminates abnormally.
- E. Program terminates abruptly.

Answer: E





```
import java.io.FileNotFoundException;  
public class Test {  
    public static void main(String[] args) {
```

```
        try {
```

```
            System.out.println(1);//Exception is not generated
```

```
        } catch (NullPointerException ex) {
```

```
            System.out.println("ONE");
```

```
        } catch (FileNotFoundException ex) { //Fully Checked Exception
```

```
            System.out.println("TWO");
```

```
        }
```

```
        System.out.println("THREE");
```

```
    }  
}
```

A. ONE

B. TWO

C. THREE

D. None of the System.out.println() be executed

E. CompilationError

Answer: E

I

46



## Answer:E

## Question 18:

Consider codes of 3 java files:

```
//Class1.java
import java.io.FileNotFoundException;
public class Class1 {
    public void read() throws FileNotFoundException {}//valid
}

//Class2.java
public class Class2 {}//valid
String Class2;//valid
public void Class2() {}//it is a normal method
}

//Class3.java
public class Class3 {}//invalid
private void print() {
    private String msg = "HELLO";//inside a method only modifier permitted is "final"
    System.out.println(msg);
}
```

47



File Edit View

```
//Class2.java
public class Class2 { //valid
    String Class2; //valid
    public void Class2() {} //it is a normal method
}
```

```
//Class3.java
public class Class3 { //invalid
    private void print() {
        private String msg = "HELLO"; //inside a method only modifier permitted is "final"
        System.out.println(msg);
    }
}
```

Which of the following statement is true?

- A. Only Class1.java compiles successfully
  - B. Only Class2.java compiles successfully
  - C. Only Class3.java compiles successfully
  - D. Class1.java and Class2.java compiles successfully
  - E. Class1.java and Class3.java compiles successfully
  - F. Class2.java and Class3.java compiles successfully
- Answer: D



Editor - (D:\Wrapper classes\Test.java)

File Edit View Search Document Project Tools Browser Emmet Window Help

Directory Object Functions

D:\New Volume

Wrapper classes

Item Test.java  
Test.java

1 import java.util.\*;

2

3 class Test

4 {

5     public static void main(String[] args)

6     {

7         //working with generic code(JDK1.5V0

8         ArrayList<String> al = new ArrayList<String>();

9         al.add("sachin");

10

11

12

13

14

15         //working with Non-generic code(jdk1.4V)

16         public void m1(ArrayList al){

17             al.add(10);

18             al.add("sachin");

19             al.add(true);

20

21         }

22     }

Test.java

for help, press F1

22°C

Priority security

Q Search

in 9

cod 23

22

22

PC

ANSI

ENG

IN

2106

20-12-2022

Man Manjanna

## D. NOT THROWING ANY EXCEPTION

Question 20:

Consider below code:

```
public class Test {  
    static {  
        System.out.println(1/0);  
    }  
  
    public static void main(String[] args) {  
        System.out.println("HELLO");  
    }  
}
```

On execution, does Test class print "HELLO" on to the console?

A. HELLO is printed on the console

B. NO Hello is not printed on the console

Answer: java.lang.[ExceptionInInitializerError](#)

B





Q> What will be the result of compiling and executing Test class?

```
public class Test {  
    public static void main(String[] args) {  
        m1();//Line 3  
    }  
}
```

```
private static void m1() throws Exception { //Line 6  
    System.out.println("NOT THROWING ANY EXCEPTION"); //Line 7  
}
```

- A. CompilationError at Line 3
- B. CompilationError at Line 7
- C. CompilationError at Line 6
- D. NOT THROWING ANY EXCEPTION

throws====> compiler check for handling code

- a. throws
- b. try and catch

OS

