

Lecture 4 - Sep. 19

Review on OOP

*Tracing OO Programs, Aliasing, Arrays
Attributes/Parameters/Return Types
Anonymous Objects*

Announcements

1. deadline
2. prog. req.

- Lab1 released (scheduled lab sessions & office hours)
- Lab0 Part 2 Due on Friday
- WrittenTest1
- (**make sure** you try logging into **eClass** in WSC)
- ProgTest1

Exercise

```

1 Person alan = new Person("Alan");
2 Person mark = new Person("Mark");
3 Person tom = new Person("Tom");
4 Person jim = new Person("Jim");
5 Person[] persons1 = {alan, mark, tom}; name  
age
6 Person[] persons2 = new Person[persons1.length];
7 for(int i = 0; i < persons1.length; i++) {
8     persons2[i] = persons1[i]; 3
9 }
10 persons1[0].setAge(70);
11 System.out.println(jim.getAge());
12 System.out.println(alan.getAge());
13 System.out.println(persons2[0].getAge());
14 persons1[0] = jim;
15 persons1[0].setAge(75);
16 System.out.println(jim.getAge());
17 System.out.println(alan.getAge());
18 System.out.println(persons2[0].getAge());

```

Person[]

persons1;

→ starts the beginning address of the array

each index of the array stores the address of some Person object

name
age

3

2

Iterations

Assertion
(alan,
70)

5
6

7

✓

persons1;

alan

Person

1. "Alan"

2. 0

Person

1. "Mark"

2. 0

Person

1. "Tom"

2. 0

Person

1. "Jim"

2. 0

alan ==
persons1[0]

array

Person2

atrasig: alan

1st iteration

persons1[0]

persons2[0]

null

null

null

null

null

null

null

null

null

0

1

2

3

4

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

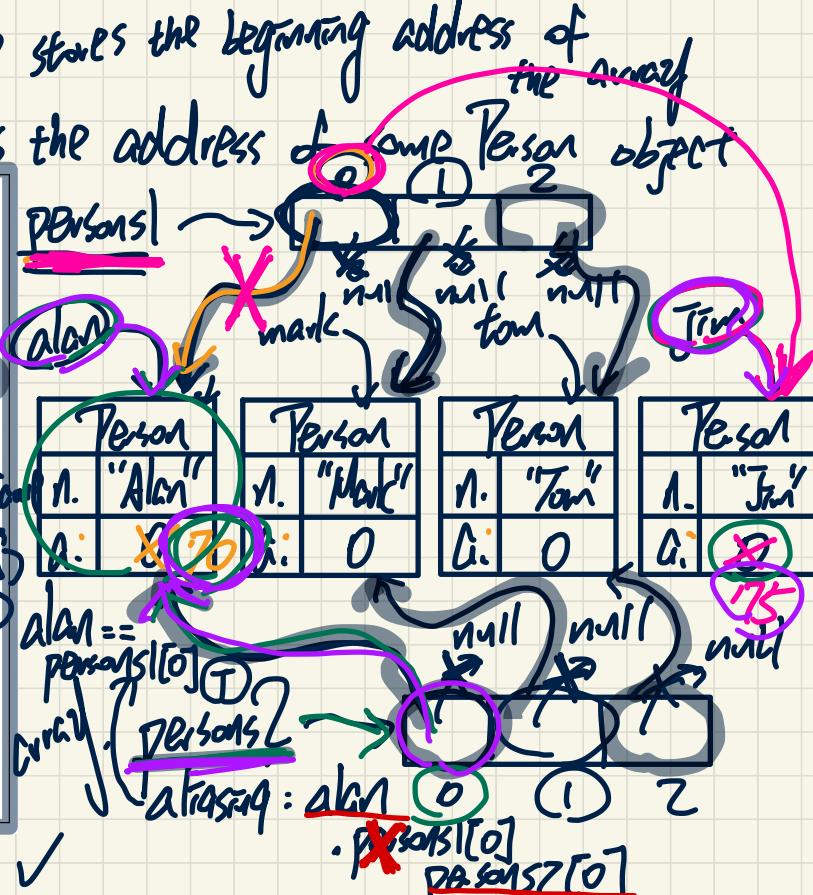
<p

Exercise

```
1 Person alan = new Person("Alan");
2 Person mark = new Person("Mark");
3 Person tom = new Person("Tom");
4 Person jim = new Person("Jim");
5 Person[] persons1 = {alan, mark, tom};           name
6 Person[] persons2 = new Person[persons1.length];   age
7 for(int i = 0; i < persons1.length; i++) {
8     persons2[i] = persons1[i];
9 }
10 persons1[0].setAge(70);
11 System.out.println(jim.getAge());                Iterations
12 System.out.println(alan.getAge());                assert
13 System.out.println(persons2[0].getAge());
14 persons1[0] = jim;
15 persons1[0].setAge(75);                         assertions
16 System.out.println(jim.getAge());                (alan,
17 System.out.println(alan.getAge());                mark, tom)
18 System.out.println(persons2[0].getAge())
```

Person[] persons = new Person[3]; ✓

→ persons[0] = alan; // copy add. stored in
→ persons[1] = mark; alan to index 0.
→ persons[2] = tom;



alan.toString() → address

Accessors/Getters vs. Mutators/Setters

```
public class Person {  
    /*  
     * Attributes.  
     * Person instances have the same attribute names.  
     * Person instances have specific attribute values.  
     */  
  
    double weight;  
    double height;  
  
    /* Accessors/Getters */  
    public double getBMI() {  
        double bmi = this.weight / (this.height * this.height);  
        return bmi;  
    }  
  
    /* Mutators/Setters */  
    public void gainWeightBy(double amount) {  
        this.weight = this.weight + amount;  
    }  
}
```

Annotations: Handwritten labels 'Jim' and 'Jon.' are placed near the BMI calculation and weight modification methods respectively.

A diagram showing a table for a 'Person' object named 'Jim'. The table has two rows: 'w.' and 'h.'. The 'w.' row contains the value '72' with a red 'X' over it and '75' written next to it. The 'h.' row contains the value '1.81' with a green checkmark over it.

| Person | |
|--------|------|
| w. | 72 |
| h. | 1.81 |

A diagram showing a table for a 'Person' object named 'Jonathan'. The table has two rows: 'w.' and 'h.'. The 'w.' row contains the value '65' with a red 'X' over it and '67' written next to it. The 'h.' row contains the value '1.67' with a pink checkmark over it.

| Person | |
|--------|------|
| w. | 65 |
| h. | 1.67 |

```
@Test  
public void test_3() {  
    Person jim = new Person(72, 1.81);  
    Person jonathan = new Person(65, 1.67);  
  
    assertEquals(21.977, jim.getBMI(), 0.01);  
    assertEquals(23.307, jonathan.getBMI(), 0.01);  
  
    jim.gainWeightBy(3);  
    jonathan.gainWeightBy(3);  
  
    assertEquals(22.893, jim.getBMI(), 0.01);  
    assertEquals(24.382, jonathan.getBMI(), 0.01);  
}
```

Object Oriented Programming (OOP)

- Templates (compile-time Java classes)
 - + attributes (common around instances)
 - + methods
 - * constructors
 - * accessors/getters
 - * mutators/setters
 - + Eclipse: Refactoring
- Instances/Entities (runtime objects)
 - + instance-specific attribute values
 - + calling constructor to create objects
 - + using the “dot notation”, with the right contexts, to:
 - * get attribute values
 - * call accessors or mutators

Use of Accessors vs. Mutators

Slide 48

```
class Person {  
    void setWeight(double weight) { ... }  
    double getBMI() { ... }  
}
```

intend to use as the argument value
in the mutator call

- Calls to **mutator methods** *cannot* be used as values.
 - ① e.g., System.out.println(jim.setWeight(78.5)); *void* X
 - ② e.g., double w = jim.setWeight(78.5); void X
 - ③ e.g., jim.setWeight(78.5); ✓
- Calls to **accessor methods** *should* be used as values.
 - ④ ✓ e.g., jim.getBMI(); ✓
 - ⑤ ✓ e.g., System.out.println(jim.getBMI()); ✓
 - ⑥ e.g., double w = jim.getBMI(); ✓



Method Parameters

mainly for
private helper
methods

Slide 49

- **Principle 1:** A **constructor** needs an **input parameter** for every attribute that you wish to initialize.

e.g., `Person(double w, double h)` vs.
`Person(String fName, String lName)`

- **Principle 2:** A **mutator** method needs an **input parameter** for every attribute that you wish to modify.

e.g., In `Point`, `void moveToXAxis()` vs.
`void moveUpBy(double unit)`

- **Principle 3:** An **accessor method** needs **input parameters** if the attributes alone are not sufficient for the intended computation to complete.

e.g., In `Point`, `double getDistFromOrigin()` vs.
`double getDistFrom(Point other)`

pl.getDFOC(); pl.getDF(P2);

Reference-Typed Return Values

Slide 53

class MyClass {

 att;
 ~~typ~~

1. primitive
2. ref type
 ↳ single-valued
 ↳ mult-valued (array)

```
public class Point {
    public void moveUpBy(int i) { y = y + i; }
    Point movedUpBy(int x, int y, Point p1) {
        Point np = new Point(x, y);
        np.moveUp(i);
        return np;
    }
}
```

- does not modify this
- modify some local var.

class Person {

Person spouse;

Person[] children;

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳

↳</

Anonymous Objects

Slide 56 - 58

```
1 double square(double x) {  
2     double sqr = x * x;  
3     return sqr; } Name
```

```
1 double square(double x) {  
2     return x * x; }
```

Anonymous exp.

```
1 Person getP(String n) { ✓  
2     Person p = new Person(n);  
3     return p; } Name
```

```
1 Person getP(String n) {  
2     return new Person(n); }
```

Anonymous
obj.

LabO P2

```
class Member {  
    private Order[] orders;  
    private int noo;  
    /* constructor omitted */  
    public void addOrder(Order o) {  
        this.orders[this.noo] = o;  
        this.noo++;  
    }  
    public void addOrder(String n, double p, double q) {  
        Order o = new Order(n, p, q);  
        this.orders[this.noo] = o;  
        this.noo++;  
    }  
}
```

overloading - Exercise

treat this
as a helper
method.

this.addOrder(5);

fh7s.addOrder
(new Order
(---, ---, ---));

Order o = new Order(n, p, q);
this.orders[this.noo] = o;
this.noo++;

dup.