

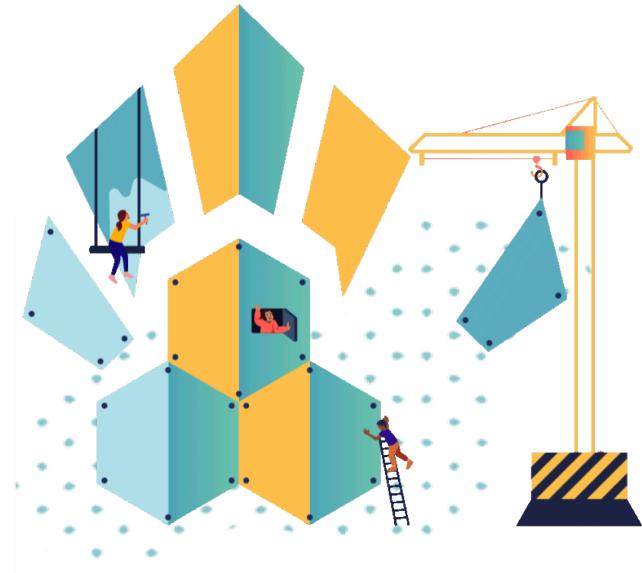
Hack@Brown 2020



Come to our info session:
<https://tinyurl.com/info2020>

And apply here:
<https://tinyurl.com/hackatbrown2020>

Due 9/13





WiCS

Women In Computer Science

Dedicated to improving diversity and inclusion across gender identity in CS.



Join our listserv to get updates and hear opportunities!

Email: wics@lists.cs.brown.edu

Add yourself to the listserv: <https://tinyurl.com/brownuwics>

WELCOME BACK EVENT ft. Kabob & Curry

Thursday, 9/12/19

CIT 368

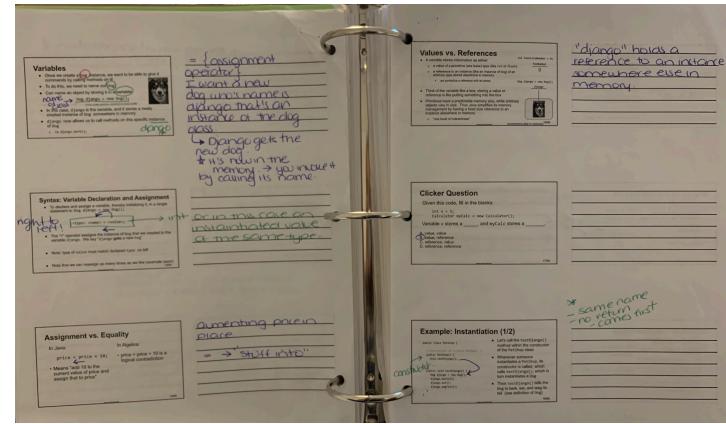
CS15 Mixer

- This Wednesday, September 11 in CIT 3rd floor atrium, from 5-6PM
- Get to know the 15 TA staff!
- Drinks and light snacks will be provided



Note Taking for CS15

- Slides are **always** uploaded to the website before lectures!
- Physical copies
 - print out the “Printable PDF” version of the slides before lecture in one of Brown’s printing centers and take notes while Andy is speaking!
 - printing center locations can be found [here!](#)
- Live note-taking
 - If you download the Power Point version of Andy’s slides, you can take notes in the lower part of the screen



My own notes from CS15!

1 Hack@Brown 2020
2 WICS
3 Note Taking for CS15
4 How to Install Top Hat
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
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
8010
8011
8012
8013
8014
8015
8016
8017
8018
8019
8020
8021
8022
8023
8024
8025
8026
8027
8028
8029
8030
8031
8032
8033
8034
8035
8036
8037
8038
8039
8040
8041
8042
8043
8044
8045
8046
8047
8048
8049
8050
8051
8052
8053
8054
8055
8056
8057
8058
8059
8060
8061
8062
8063
8064
8065
8066
8067
8068
8069
8070
8071
8072
8073
8074
8075
8076
8077
8078
8079
8080
8081
8082
8083
8084
8085
8086
8087
8088
8089
8090
8091
8092
8093
8094
8095
8096
8097
8098
8099
80100
80101
80102
80103
80104
80105
80106
80107
80108
80109
80110
80111
80112
80113
80114
80115
80116
80117
80118
80119
80120
80121
80122
80123
80124
80125
80126
80127
80128
80129
80130
80131
80132
80133
80134
80135
80136
80137
80138
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80199
80200
80201
80202
80203
80204
80205
80206
80207
80208
80209
80210
80211
80212
80213
80214
80215
80216
80217
80218
80219
80220
80221
80222
80223
80224
80225
80226
80227
80228
80229
80230
80231
80232
80233
80234
80235
80236
80237
80238
80239
80240
80241
80242
80243
80244
80245
80246
80247
80248
80249
80250
80251
80252
80253
80254
80255
80256
80257
80258
80259
80260
80261
80262
80263
80264
80265
80266
80267
80268
80269
80270
80271
80272
80273
80274
80275
80276
80277
80278
80279
80280
80281
80282
80283
80284
80285
80286
80287
80288
80289
80290
80291
80292
80293
80294
80295
80296
80297
80298
80299
80300
80301
80302
80303
80304
80305
80306
80307
80308
80309
80310
80311
80312
80313
80314
80315
80316
80317
80318
80319
80320
80321
80322
80323
80324
80325
80326
80327
80328
80329
80330
80331
80332
80333
80334
80335
80336
80337
80338
80339
80340
80341
80342
80343
80344
80345
80346
80347
80348
80349
80350
80351
80352
80353
80354
80355
80356
80357
80358
80359
80360
80361
80362
80363
80364
80365
80366
80367
80368
80369
80370
80371
80372
80373
80374
80375
80376
80377
80378
80379
80380
80381
80382
80383
80384
80385
80386
80387
80388
80389
80390
80391
80392
80393
80394
80395
80396
80397
80398
80399
80400
80401
80402
80403
80404
80405
80406
80407
80408
80409
80410
80411
80412
80413
80414
80415
80416
80417
80418
80419
80420
80421
80422
80423
80424
80425
80426
80427
80428
80429
80430
80431
80432
80433
80434
80435
80436
80437
80438
80439
80440
80441
80442
80443
80444
80445
80446
80447
80448
80449
80450
80451
80452
80453
80454
80455
80456
80457
80458
80459
80460
80461
80462
80463
80464
80465
80466
80467
80468
80469
80470
80471
80472
80473
80474
80475
80476
80477
80478
80479
80480
80481
80482
80483
80484
80485
80486
80487
80488
80489
80490
80491
80492
80493
80494
80495
80496
80497
80498
80499
80500
80501
80502
80503
80504
80505
80506
80507
80508
80509
80510
80511
80512
80513
80514
80515
80516
80517
80518
80519
80520
80521
80522
80523
80524
80525
80526
80527
80528
80529
80530
80531
80532
80533
80534
80535
80536
80537
80538
80539
80540
80541
80542
80543
80544
80545
80546
80547
80548
80549
80550
80551
80552
80553
80554
80555
80556
80557
80558
80559
80560
80561
80562
80563
80564
80565
80566
80567
80568
80569
80570
80571
80572
80573
80574
80575
80576
80577
80578
80579
80580
80581
80582
80583
80584
80585
80586
80587
80588
80589
80590
80591
80592
80593
80594
80595
80596
80597
80598
80599
80600
80601
80602
80603
80604
80605
80606
80607
80608
80609
80610
80611
80612
80613
80614
80615
80616
80617
80618
80619
80620
80621
80622
80623
80624
80625
80626
80627
80628
80629
80630
80631
80632
80633
80634
80635
80636
80637
80638
80639
80640
80641
80642
80643
80644
80645
80646
80647
80648
80649
80650
80651
80652
80653
80654
80655
80656
80657
80658
80659
80660
80661
80662
80663
80664
80665
80666
80667
80668
80669
80670
80671
80672
80673
80674
80675
80676
80677
80678
80679
80680
80681
80682
80683
80684
80685
80686
80687
80688
80689
80690
80691
80692
80693
80694
80695
80696
80697
80698
80699
80700
80701
80702
80703
80704
80705
80706
80707
80708
80709
80710
80711
80712
80713
80714
80715
80716
80717
80718
80719
80720
80721
80722
80723
80724
80725
80726
80727
80728
80729
80730
80731
80732
80733
80734
80735
80736
80737
80738
80739
80740
80741
80742
80743
80744
80745
80746
80747
80748
80749
80750
80751
80752
80753
80754
80755
80756
80757
80758
80759
80760
80761
80762
80763
80764
80765
80766
80767
80768
80769
80770
80771
80772
80773
80774
80775
80776
80777
80778
80779
80780
80781
80782
80783
80784
80785
80786
80787
80788
80789
80790
80791
80792
80793
80794
80795
80796
80797
80798
80799
80800
80801
80802
80803
80804
80805
80806
80807
80808
80809
80810
80811
80812
80813
80814
80815
80816
80817
80818
80819
80820
80821
80822
80823
80824
80825
80826
80827
80828
80829
80830
80831
80832
80833
80834
80835
80836
80837
80838
80839
80840
80841
80842
80843
80844
80845
80846
80847
80848
80849
80850
80851
80852
80853
80854
80855
80856
80857
80858
80859
80860
80861
80862
80863
80864
80865
80866
80867
80868
80869
80870
80871
80872
80873
80874
80875
80876
80877
80878
80879
80880
80881
80882
80883
80884
80885
80886
80887
80888
80889
80890
80891
80892
80893
80894
80895
80896
80897
80898
80899
80900
80901
80902
80903
80904
80905
80906
80907
80908
80909
80910
80911
80912
80913
80914
80915
80916
80917
80918
80919
80920
80921
80922
80923
80924
80925
80926
80927
80928
80929
80930
80931
80932
80933
80934
80935
80936
80937
80938
80939
80940
80941
80942
80943
80944
80945
80946
80947
80948
80949
80950
80951
80952
80953
80954
80955
80956
80957
80958
80959
80960
80961
80962
80963
80964
80965
80966
80967
80968
80969
80970
80971
80972
80973
80974
80975
80976
80977
80978
80979
80980
80981
80982
80983
80984
80985
80986
80987
80988
80989
80990
80991
80992
80993
80994
80995
80996
80997
80998
80999
80100
80101
80102
80103
80104
80105
80106
80107
80108
80109
80110
80111
80112
80113
80114
80115
80116
80117
80118
80119
80120
80121
80122
80123
80124
80125
80126
80127
80128
80129
80130
80131
80132
80133
80134
80135
80136
80137
80138
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80199
80200
80201
80202
80203
80204
80205
80206
80207
80208
80209
80210
80211
80212
80213
80214
80215
80216
80217
80218
80219
80220
80221
80222
80223
80224
80225
80226
80227
80228
80229
80230
80231
80232
80233
80234
80235
80236
80237
80238
80239
80240
80241
80242
802

How to Install TopHat

TOP HAT

- Computer: Go to <https://tophat.com> → Click *Signup* in upper right corner → select *Student* → join with course code or *Search by School* → input info under *Account* → enter your Banner ID under *Grading* → Add your phone number to submit responses in class via text under *Phone*
- IOS/Android: Download **Top Hat Lecture** App → click *Create Student Account* and follow instructions to complete
- Link with Detailed Instructions: <https://tinyurl.com/y6ythebb>
- CS15 Course Code: 783865

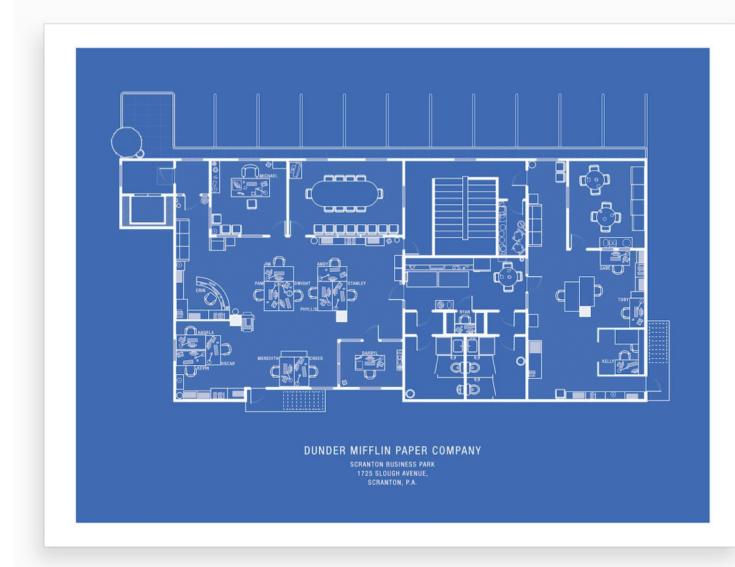


Review

- We model the “application world” as a system of collaborating objects
- Objects collaborate by sending each other messages
- Objects have properties and behaviors (things they know how to do)
- Objects are typically composed of component objects

Lecture 2

Calling and Defining Methods in Java



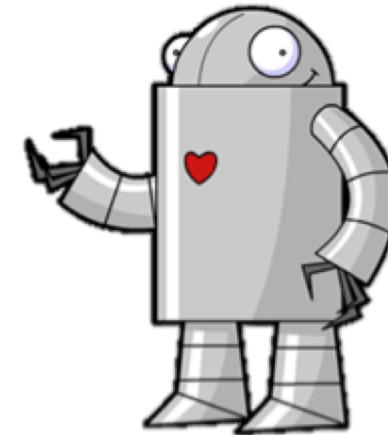
Outline

- Calling methods
- Declaring and defining a class
- Instances of a class
- Defining methods
- The this keyword

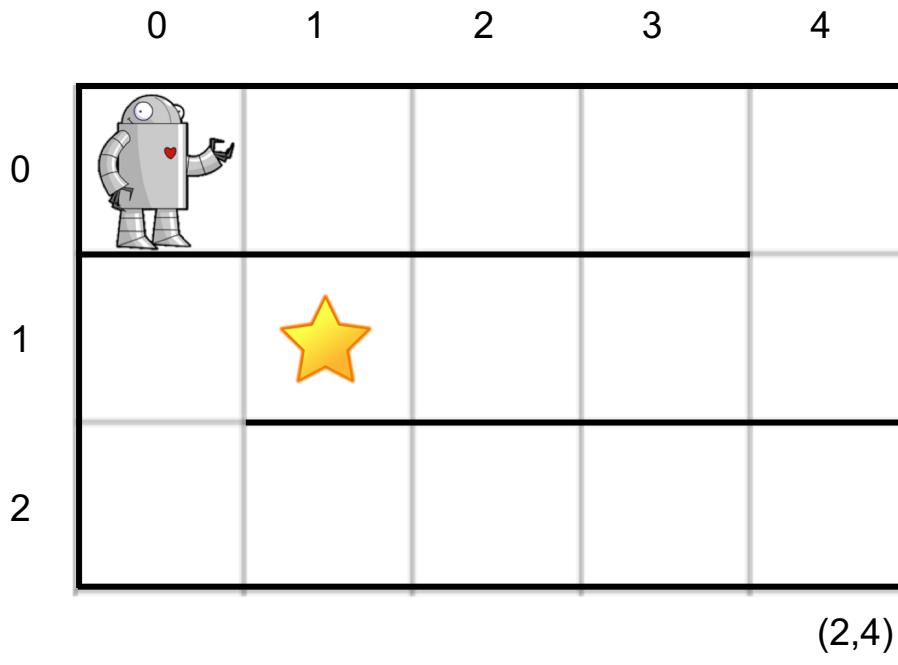
Meet samBot

(kudos to former headTA Sam Squires)

- **samBot** is a robot who lives in a 2D grid world
- She knows how to do two things:
 - move forward any number of steps
 - turn right 90°
- We will learn how to communicate with **samBot** using Java



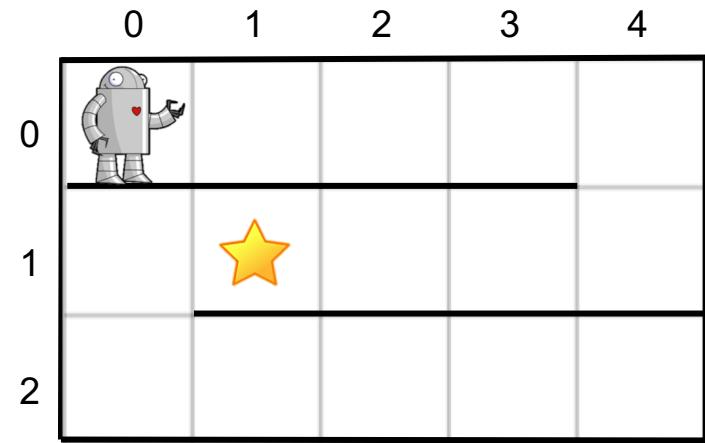
samBot's World



- This is **samBot**'s world
- **samBot** starts in the square at (0,0)
- She wants to get to the square at (1,1)
- Thick black lines are walls **samBot** can't pass through

Giving Instructions (1/3)

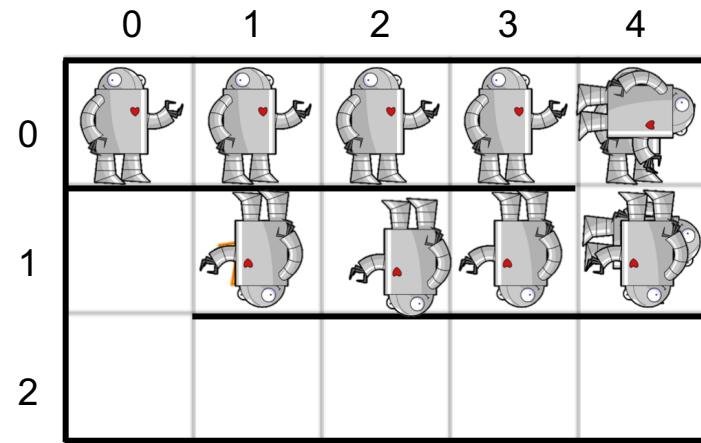
- **Goal:** move **samBot** from starting position to destination by giving her a list of instructions
- **samBot** only knows how to “move forward n steps” and “turn right”
- What instructions should be given?



Giving Instructions (2/3)

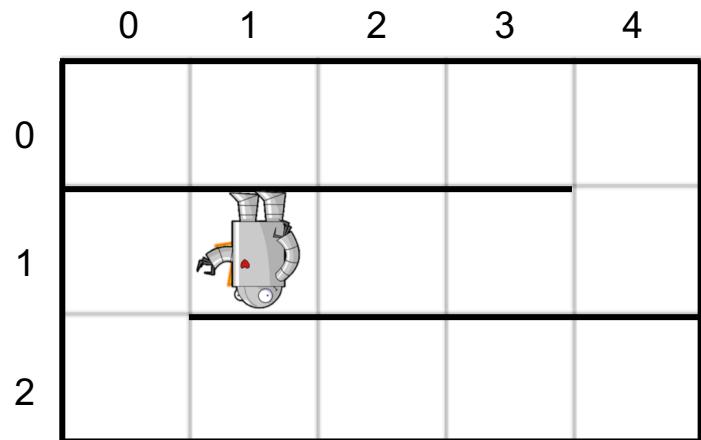
Note: samBot moves in the direction her outstretched arm is pointing.
Yes, she can move sideways and upside down in this 2D world!

- “Move forward 4 steps.”
- “Turn right.”
- “Move forward 1 step.”
- “Turn right.”
- “Move forward 3 steps.”



Giving Instructions (3/3)

- Instructions have to be given in a language `samBot` knows
- That's where Java comes in!
- In Java, give instructions to an object by **giving it commands**

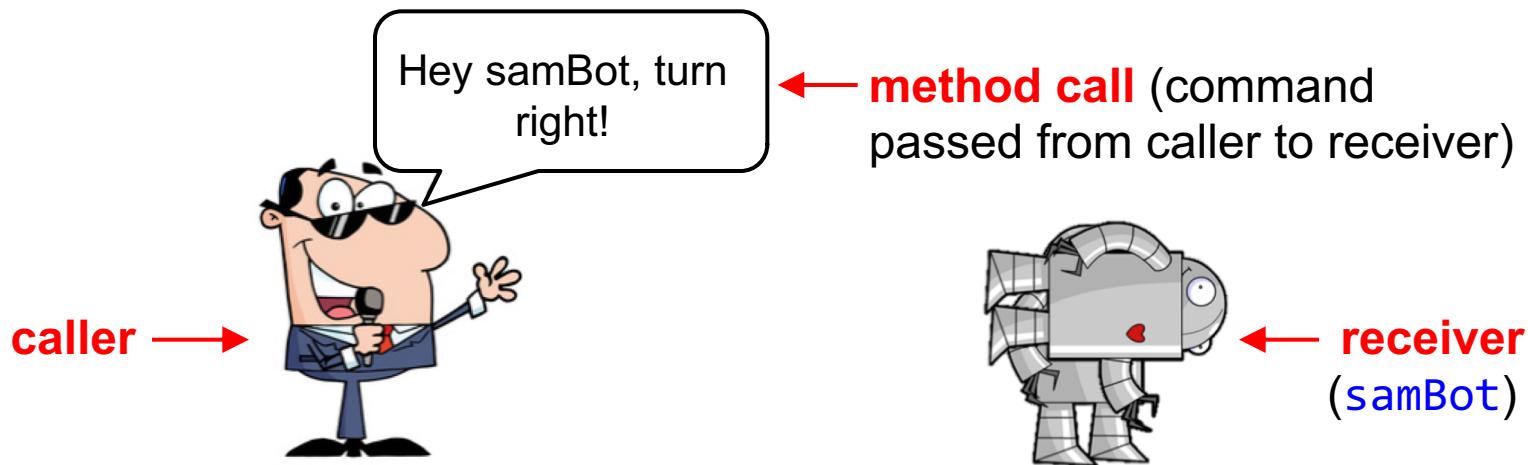


“Calling Methods”: Giving Commands in Java (1/2)

- **samBot** can only handle commands she knows how to respond to
- These responses are called **methods!**
 - “method” is short for “method for responding to a command”. Therefore, whenever **samBot** gets a command, she can respond by utilizing a method.
- Objects cooperate by giving each other commands
 - **caller** is the object giving the command
 - **receiver** is the object receiving the command

“Calling Methods”: Giving Commands in Java (2/2)

- `samBot` already has one method for “move forward n steps” and another method for “turn right”
- When we send a command to `samBot` to “move forward” or “turn right” in Java, we are **calling a method on `samBot`**.



Turning samBot right

Names don't have spaces!
Style guide has capitalization conventions, e.g., camelCase

- `samBot`'s “turn right” method is called `turnRight`
- To call the `turnRight` method on `samBot`:

```
samBot.turnRight();
```

- To call methods on `samBot` in Java, need to address her by name!
- Every command to `samBot` takes the form:

```
samBot.<method name(...)>;
```

You can substitute anything in < >!

; ends Java statement

- What are those parentheses at the end of the method for?

Moving samBot forward

- Remember: when telling `samBot` to move forward, you need to tell her how many steps to move
- `samBot`'s “move forward” method is named `moveForward`
- To call this method in Java:

```
    samBot.moveForward(<number of steps>);
```

- This means that if we want her to move forward 2 steps, we say:

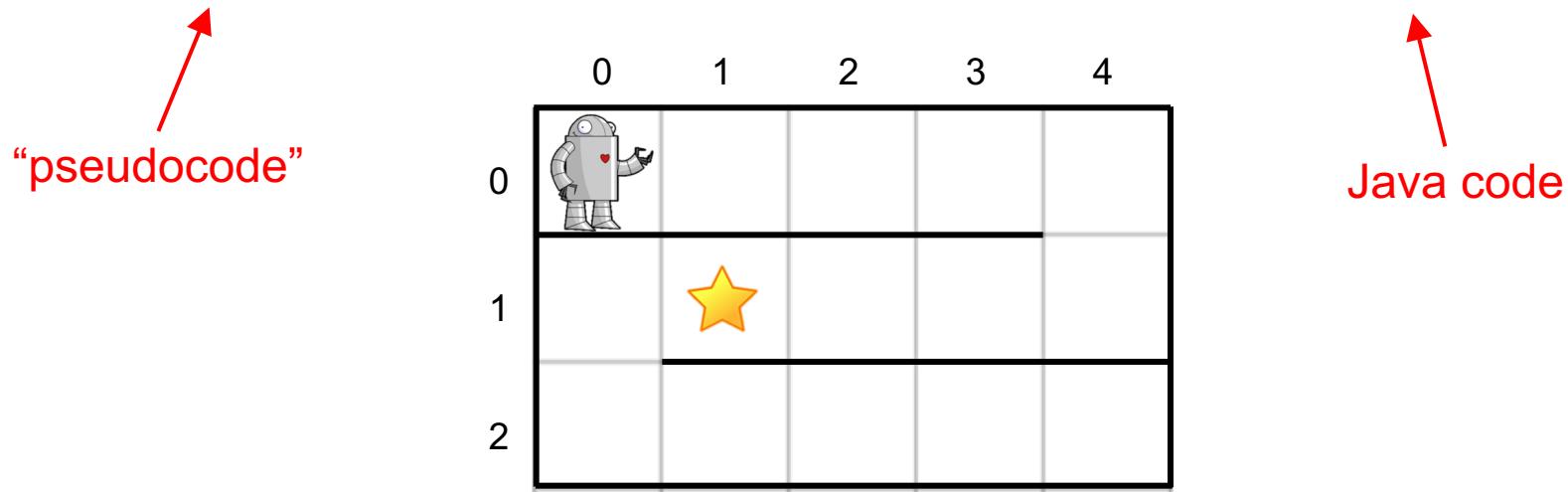
```
    samBot.moveForward(2);
```

Calling Methods: Important Points

- Method calls in Java have parentheses after the method's name
- In the **definition** of the method, extra pieces of information to be passed into the method are called **parameters**; in the **call** to the method, the actual values passed in are called **arguments**
 - e.g. : in **defining** `f(x)`, `x` is the parameter; in **calling** `f(2)`, `2` is the argument
 - more on parameters and arguments next lecture!
- If the method needs any information, include it between the parentheses (e.g., `samBot.moveForward(2);`)
- If no extra information is needed, just leave the parentheses empty (e.g., `samBot.turnRight();`)

Guiding samBot in Java

- Tell `samBot` to move forward 4 steps → `samBot.moveForward(4);`
- Tell `samBot` to turn right → `samBot.turnRight();`
- Tell `samBot` to move forward 1 step → `samBot.moveForward(1);`
- Tell `samBot` to turn right → `samBot.turnRight();`
- Tell `samBot` to move forward 3 steps → `samBot.moveForward(3);`



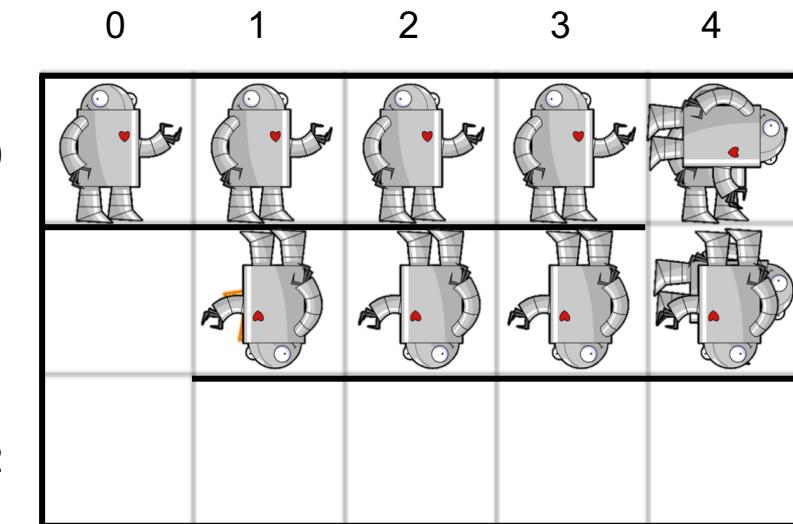
Hand Simulation

- Simulating lines of code **by hand** checks that each line produces correct action
 - we did this in slide 7 for pseudocode
- In **hand simulation**, you play the role of the computer
 - lines of code are “instructions” for the computer
 - try to follow “instructions” and see if you get desired result
 - if result is incorrect:
 - one or more instructions or the order of instructions may be incorrect



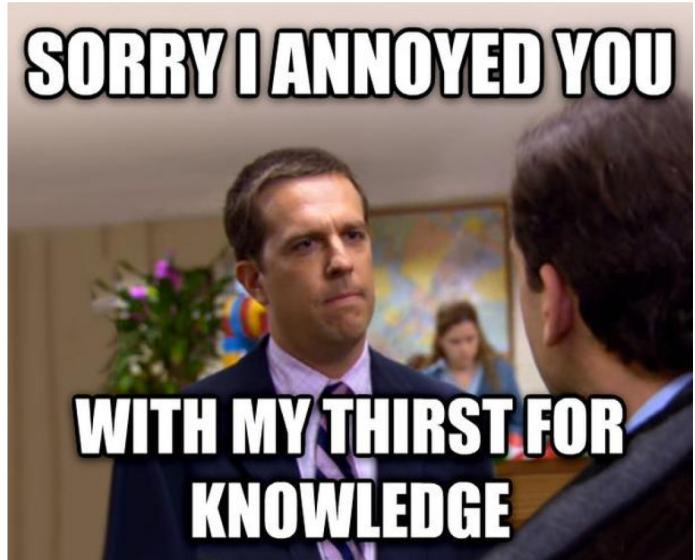
Hand Simulation of This Code

```
samBot.moveForward(4);  
samBot.turnRight();  
samBot.moveForward(1);  
samBot.turnRight();  
samBot.moveForward(3);
```



About TopHat Questions

- Increase engagement during lecture!
- We encourage working with a neighbor and discussing concepts on all TopHat questions
- Can use app, website
 - If you need a device to access TopHat, you can borrow a laptop from the IT Service Center on 5th floor of Page-Robinson Hall.



- TopHat questions are worth 5% of your grade! (See course missive)

TopHat Question

Where will `samBot` end up when this code is executed?

```
samBot.moveForward(3);  
samBot.turnRight();  
samBot.turnRight();  
samBot.moveForward(1);
```

Choose one of the positions or
E: None of the above

	0	1	2	3	4
0					B
1					C
2					
3				D	
4					

Putting Code Fragments in a Real Program (1/2)

- Let's demonstrate this code for real
- First, put it inside real Java program
- Grayed-out code specifies context in which an arbitrary robot named **myRobot** executes instructions
 - it is part of the **stencil code** written for you by the TAs, which also includes **samBot**'s or any other robot's capability to respond to **moveForward** and **turnRight**– more on this later

```
public class RobotMover {  
    /* additional stencil code elided*/  
  
    public void moveRobot(Robot myRobot) {  
        myRobot.moveForward(4);  
        myRobot.turnRight();  
        myRobot.moveForward(1);  
        myRobot.turnRight();  
        myRobot.moveForward(3);  
    }  
}
```

comment

Putting Code Fragments in a Real Program (2/2)

- Before, we've talked about objects that handle messages with "methods"
- Introducing a new concept... **classes!**

```
public class RobotMover {  
    /* additional code elided */  
  
    public void moveRobot(Robot myRobot) {  
        myRobot.moveForward(4);  
        myRobot.turnRight();  
        myRobot.moveForward(1);  
        myRobot.turnRight();  
        myRobot.moveForward(3);  
    }  
}
```

We're about to explain this part of the code!

What is a class?

- A **class** is a **blueprint** for a certain type of object
- An object's class defines its properties and capabilities (methods)
 - more on this in a few slides!
- Let's embed the **moveRobot** code fragment (method) that moves **samBot** (or any other **Robot** instance) in a new class called **RobotMover**
- Need to tell Java compiler about **RobotMover** before we can use it

```
public class RobotMover {  
    /* additional code elided */  
  
    public void moveRobot(Robot myRobot) {  
        myRobot.moveForward(4);  
        myRobot.turnRight();  
        myRobot.moveForward(1);  
        myRobot.turnRight();  
        myRobot.moveForward(3);  
    }  
}
```

Declaring and Defining a Class (1/3)

- Like a dictionary entry, first **declare** term, then provide **definition**
- First line **declares** RobotMover class
- Breaking it down:
 - **public** indicates any other object can use instances of this class
 - **class** indicates to Java compiler that we are about to define a new class
 - **RobotMover** is the name we have chosen for our class

declaration of the RobotMover class



```
public class RobotMover {  
    /* additional code elided */  
  
    public void moveRobot(Robot myRobot) {  
        myRobot.moveForward(4);  
        myRobot.turnRight();  
        myRobot.moveForward(1);  
        myRobot.turnRight();  
        myRobot.moveForward(3);  
    }  
}
```

Note: **public** and **class** are Java “reserved words” aka “keywords” and have pre-defined meanings in Java; use Java keywords a lot in the future

Declaring and Defining a Class (2/3)

- **Class definition** (aka “body”) defines properties and capabilities of class
 - it is contained within curly braces that follow the class declaration
- A class’s **capabilities** (“what it knows how to do”) are defined by its **methods** – `RobotMover` thus far only shows one specific method, `moveRobot`
 - A method is a declaration followed by its body (also enclosed in {...} braces)
- A class’s **properties** are defined by its **instance variables** – more on this next week

```
public class RobotMover {
```

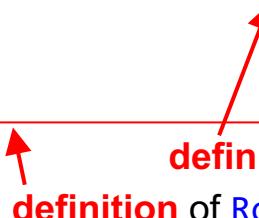
```
/* additional code elided */
```

```
public void moveRobot(Robot myRobot) {
```

```
    myRobot.moveForward(4);  
    myRobot.turnRight();  
    myRobot.moveForward(1);  
    myRobot.turnRight();  
    myRobot.moveForward(3);
```

```
}
```

```
}
```


definition of RobotMover class
definition of moveRobot method

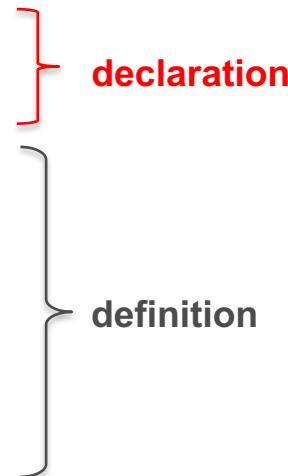
Declaring and Defining a Class (3/3)

- General form for a class:

```
<visibility> class <name> {
```

```
    <code (properties and  
    capabilities) that defines class>
```

```
}
```



declaration

definition

- to make code more compact, typically put opening brace on same line as declaration -- Java compiler doesn't care
- Each class goes in its own file, where **name of file matches name of class**
 - RobotMover class is contained in file “RobotMover.java”

The Robot class (defined by the TAs)

Note: Normally, support code is a “black box” that you can’t examine



```
public class Robot {  
  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberofSteps) {  
        // code that moves robot forward  
    }  
  
    /* other code elided-- if you're curious, check out  
    Robot.java in the stencil code!*/  
}
```

in-line comment

- **public class Robot declares** a **class** called **Robot**
- Information about the properties and capabilities of **Robots** (the **class definition**) goes within the red curly braces

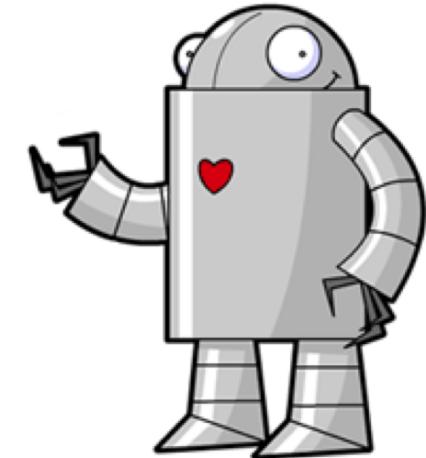
Methods of the TA's Robot class

```
public class Robot {  
  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberOfSteps) {  
        // code that moves robot forward  
    }  
  
    /* other code elided-- if you're curious, check  
    out Robot.java in the stencil code!*/  
}
```

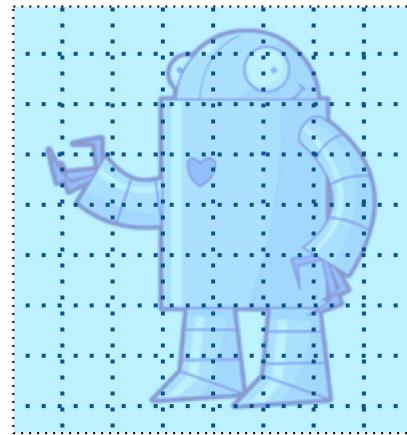
- `public void turnRight()` and `public void moveForward(int numberOfSteps)` each **declare a method**
 - more on `void` later!
- `moveForward` needs to know how many steps to move, so the parameter is `int numberOfSteps` within parentheses
 - `int` tells compiler this parameter is an “integer” (we say “`moveForward` takes a single parameter called `numberOfSteps` of type `int`”)

Classes and Instances (1/4)

- `samBot` is an **instance** of class `Robot`
 - this means `samBot` is a particular `Robot` that was built using the `Robot` class as a blueprint (another **instance** could be `dwightBot`)
- All `Robots` (all **instances** of the class `Robot`) have **the exact same capabilities**: the methods defined in the `Robot` class. What one `Robot` **instance** can do, they all can do since they are made with the same blueprint!
- All `Robots` also have **the exact same properties** (i.e., every `Robot` has a `Color` and a `Size`)
 - they all have these properties but the values of these properties may differ between instances (e.g., a big `samBot` and small `dwightBot`)



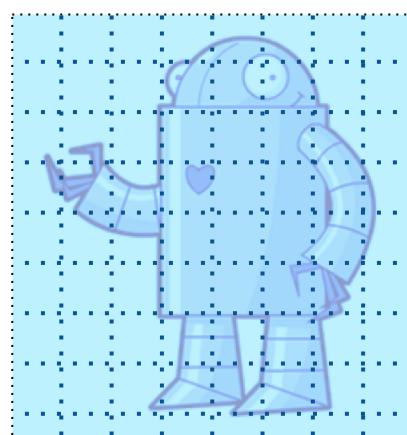
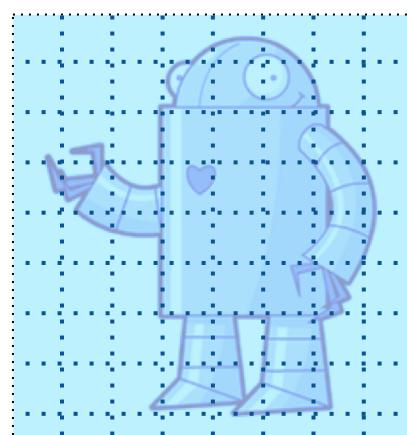
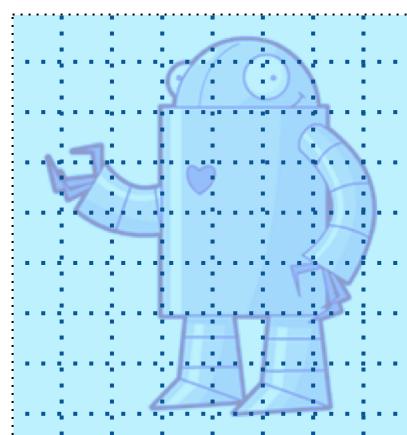
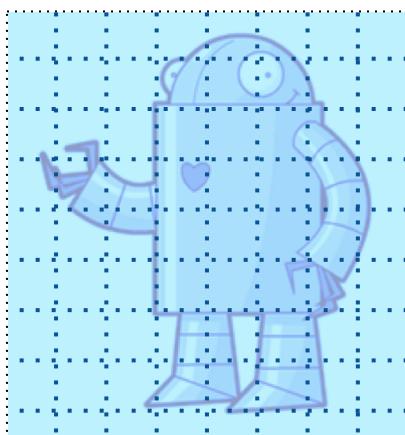
Classes and Instances (2/4)



The [Robot](#) class is
like a [blueprint](#)

Classes and Instances (3/4)

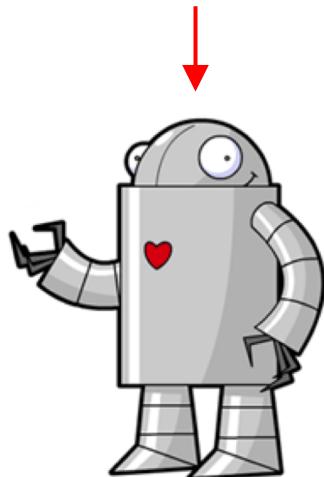
We can use the `Robot` class to build actual `Robots` - **instances** of the class `Robot`, whose properties (like their color in this case) may vary (next lecture)



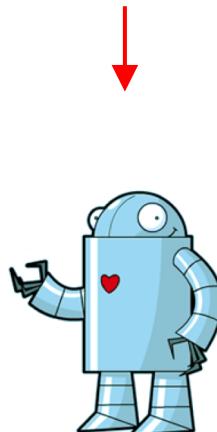
Classes and Instances (4/4)

Method calls are done on instances of the class. These are four instances of the same class (blueprint).

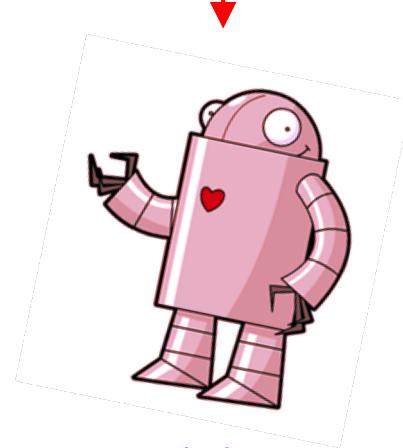
instance



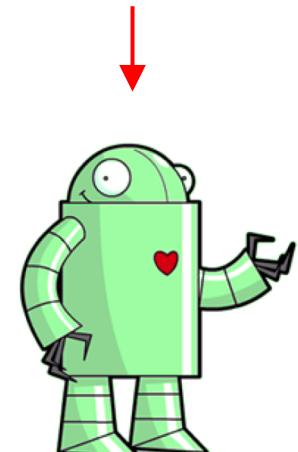
instance



instance



instance



samBot

blueBot

pinkBot

greenBot

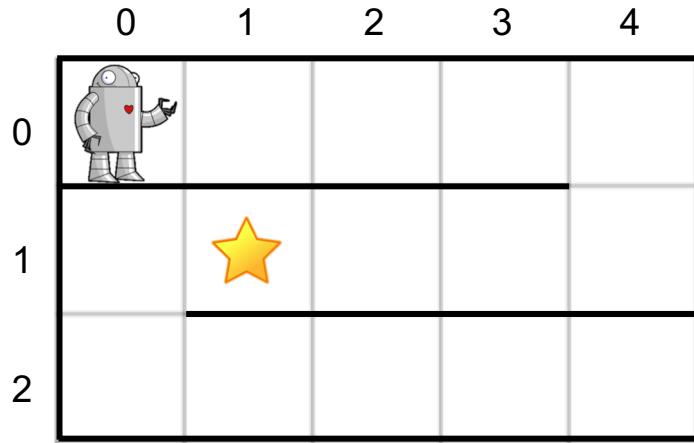
TopHat Question

You know that `blueBot` and `pinkBot` are instances of the same class. Let's say that the call

`pinkBot.chaChaSlide();` makes `pinkBot` do the cha-cha slide. Which of the following is true?

- A. The call `blueBot.chaChaSlide();` will make `blueBot` do the cha-cha slide
- B. The call `blueBot.chaChaSlide();` might make `blueBot` do the cha-cha slide or another popular line dance instead
- C. You have no guarantee that `blueBot` has the method `chaChaSlide();`

Defining Methods



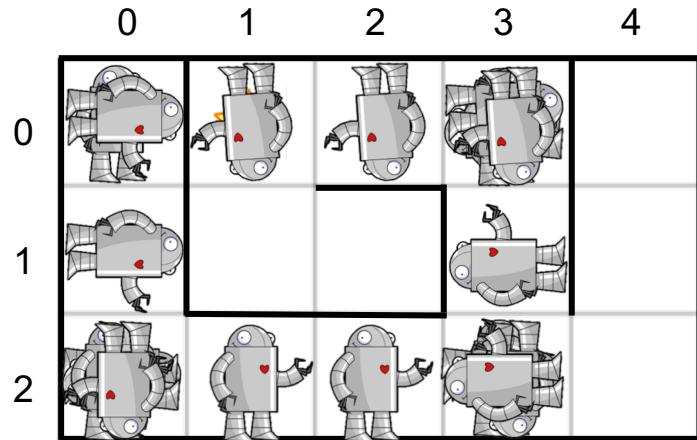
- We have already learned about **defining classes**, let's now talk about **defining methods**
- Let's use a variation of our previous example

```
public class RobotMover {  
    /* additional code elided */  
  
    public void moveRobot(Robot myRobot) {  
        // Your code goes here!  
        // ...  
        // ...  
    }  
}
```

Declaring vs. Defining Methods

- **Declaring** a method says the class knows how to do some task like `pinkBot` can `chaChaSlide()`
- **Defining** a method actually explains how the class completes this task (what command it gives) `chaChaslide()` could include: stepping backwards, alternating feet, stepping forward
- Usually you will need to both **define** and **declare** your methods

A Variation on moveRobot (1/2)



```
public class RobotMover {  
    /* additional code elided */  
  
    public void moveRobot(Robot myRobot) {  
        myRobot.turnRight();  
        myRobot.moveForward(2);  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.moveForward(3);  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.moveForward(2);  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.moveForward(2);  
    }  
}
```

A Variation on moveRobot (2/2)

- Lots of code for a simple problem..
- **samBot** only knows how to turn right, so have to call **turnRight** three times to make her turn left
- If she understood how to “turn left”, would be much less code!
- We can ask the TAs to modify **samBot** to turn left by **declaring** and **defining a method** called **turnLeft**

```
public class RobotMover {  
    /* additional code elided */  
  
    public void moveRobot(Robot myRobot) {  
        myRobot.turnRight();  
        myRobot.moveForward(2);  
        myRobot.turnRight(); } "turn left"  
        myRobot.turnRight(); } "turn left"  
        myRobot.turnRight(); } "turn left"  
        myRobot.moveForward(3);  
        myRobot.turnRight(); } "turn left"  
        myRobot.turnRight(); } "turn left"  
        myRobot.turnRight(); } "turn left"  
        myRobot.moveForward(2);  
        myRobot.turnRight(); } "turn left"  
        myRobot.turnRight(); } "turn left"  
        myRobot.turnRight(); } "turn left"  
        myRobot.moveForward(2);  
    }  
}
```

Defining a Method (1/2)

```
public class Robot {  
  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberOfSteps) {  
        // code that moves robot forward  
    }  
  
}
```

- Almost all methods take on this general form:

`<visibility> <type> <name> (<parameters>) {
 <list of statements within method>
}`
- When **calling** **turnRight** or **moveForward** on an **instance** of the **Robot** class, all code between method's curly braces is executed

Defining a Method (2/2)

```
public class Robot {  
  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberOfSteps) {  
        // code that moves robot forward  
    }  
  
    public void turnLeft() {  
        //The TA's code goes here!!  
        //Here you'll have the method definition!  
    }  
}
```

- We're going to **define** a new method: **turnLeft**
- To make a **Robot** turn left, tell her to turn right three times

The `this` keyword (1/2)

```
public class Robot {  
  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberOfSteps) {  
        // code that moves robot forward  
    }  
  
    public void turnLeft() {  
        this.turnRight();  
        this.turnRight();  
        this.turnRight();  
    }  
}
```

- When working with `RobotMover`, we were talking to `samBot`, an instance of class `Robot`
- To tell her to turn right, we said “`samBot.turnRight();`”
- Why do the TAs now write “`this.turnRight();`”?

The `this` keyword (2/2)

```
public class Robot {  
  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberOfSteps) {  
        // code that moves robot forward  
    }  
  
    public void turnLeft() {  
        this.turnRight();  
        this.turnRight();  
        this.turnRight();  
    }  
}
```

- The `this` keyword is how an instance (like `samBot`) can call a method on itself
- Use `this` to call a method of `Robot` class from within another method of the `Robot` class
- When `samBot` is told by, say, a `RobotMover` instance to `turnLeft`, she responds by telling herself to `turnRight` three times
- `this.turnRight();` means “hey me, turn right!”
- `this` is optional, but CS15 expects it

We're done!

```
public class Robot {  
  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberOfSteps) {  
        // code that moves robot forward  
    }  
  
    public void turnLeft() {  
        this.turnRight();  
        this.turnRight();  
        this.turnRight();  
    }  
}
```

- Have now seen our first method definition!
- Now that **Robot** has **turnLeft**, can call **turnLeft** on any instance of **Robot**

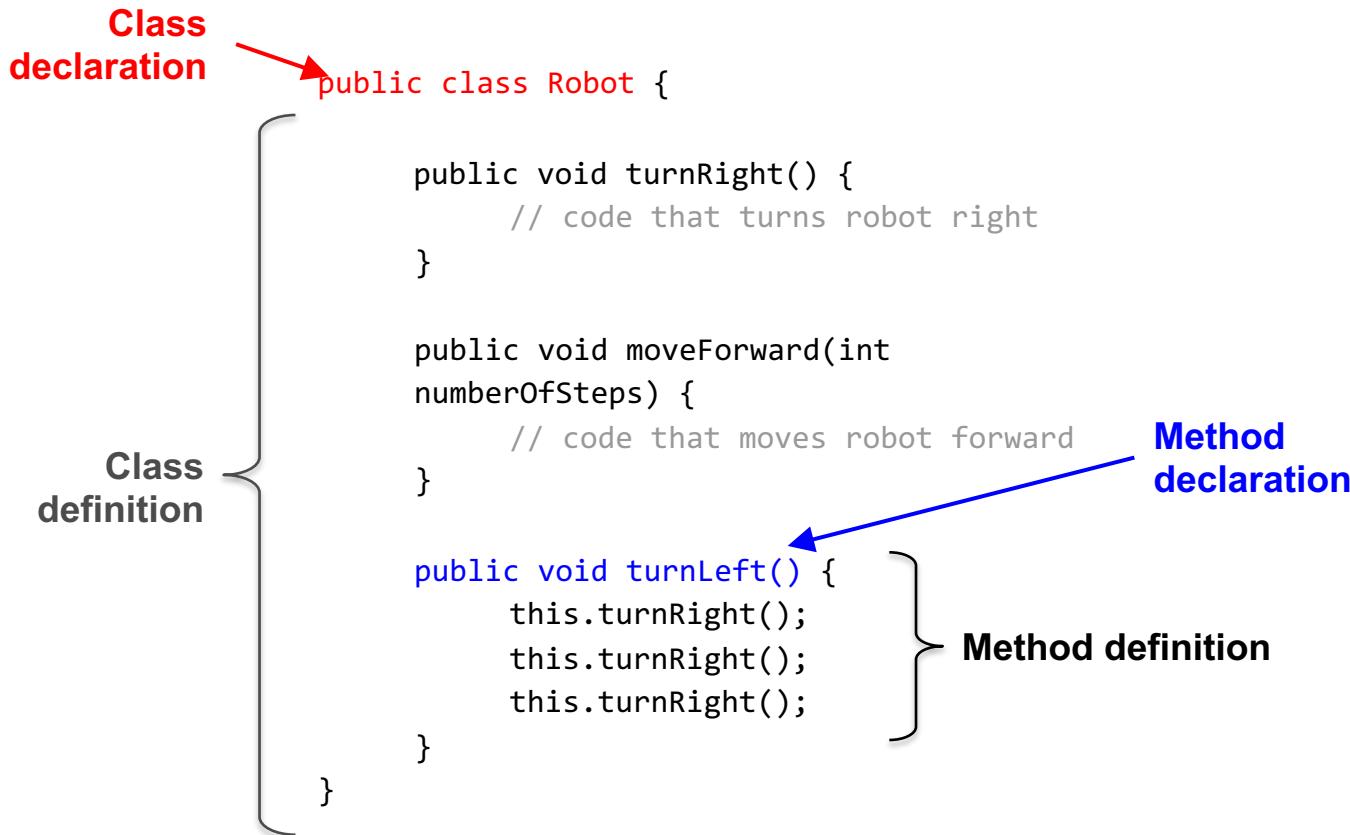
TopHat Question

```
public class Robot {  
    /* additional code elided */  
  
    public void turnLeft() {  
        this.turnRight();  
        this.turnRight();  
        this.turnRight();  
    }  
}
```

Given this method, what can we say about `this.turnRight()`?

- A. Other objects cannot call the `turnRight()` method on instances of the `Robot` class
- B. The current instance of the `Robot` class is calling `turnRight()` on another instance of `Robot`
- C. The current instance of the `Robot` class is calling the `turnRight()` method on itself
- D. The call `this.turnRight();` will not appear anywhere else in the `Robot`'s class definition

Summary



Simplifying our code using turnLeft

```
public class RobotMover {  
    public void moveRobot(Robot myRobot) {  
        myRobot.turnRight();  
        myRobot.moveForward(2);  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.moveForward(3);  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.moveForward(2);  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.turnRight();  
        myRobot.moveForward(2);  
    }  
}
```

```
public class RobotMover {  
    public void moveRobot(Robot myRobot) {  
        myRobot.turnRight();  
        myRobot.moveForward(2);  
        myRobot.turnLeft();  
        myRobot.moveForward(3);  
        myRobot.turnLeft();  
        myRobot.moveForward(2);  
        myRobot.turnLeft();  
        myRobot.moveForward(2);  
    }  
}
```

We've saved a lot of lines of code by using turnLeft!

This is good! More lines of code makes your program harder to read and more difficult to debug and maintain.

turnAround (1/3)

- The TAs could also define a method that turns the **Robot** around 180°.
- See if you can declare and define the method **turnAround**

```
public class Robot {  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int number_of_steps) {  
        // code that moves robot forward  
    }  
  
    public void turnLeft() {  
        this.turnRight();  
        this.turnRight();  
        this.turnRight();  
    }  
  
    // your code goes here!  
    // ...  
    // ...  
    // ...  
}
```

turnAround (2/3)

- Now that the `Robot` class has the method `turnAround`, we can call the method on any instance of the class `Robot`
- There are other ways of implementing this method that are just as correct

```
public class Robot {  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberOfSteps) {  
        // code that moves robot forward  
    }  
  
    public void turnLeft() {  
        this.turnRight();  
        this.turnRight();  
        this.turnRight();  
    }  
  
    public void turnAround() {  
        this.turnRight();  
        this.turnRight();  
    }  
}
```

turnAround (3/3)

- Instead of calling `turnRight`, could call our newly created method, `turnLeft`
- Both of these solutions are equally correct, in that they will turn the robot around 180°
- How do they differ? When we try each of these implementations with `samBot`, what will we see in each case?

```
public class Robot {  
    public void turnRight() {  
        // code that turns robot right  
    }  
  
    public void moveForward(int numberOfSteps) {  
        // code that moves robot forward  
    }  
  
    public void turnLeft() {  
        this.turnRight();  
        this.turnRight();  
        this.turnRight();  
    }  
  
    public void turnAround() {  
        this.turnRight();  
        this.turnRight();  
    }  
}
```

Summary (1/2)

- Classes
 - a **class** is a blueprint for a certain type of object
 - example: **Robot** is a class
- Instances
 - an **instance** of a class is a particular member of that class whose methods we can call
 - example: **samBot** is an **instance** of **Robot**

Summary (2/2)

- Calling methods
 - an instance can call on the methods defined by its class
 - **general form:** `instance.<method name>(<parameters>)`
 - example: `samBot.turnRight();`
- Defining methods
 - how we describe a capability of a class
 - **general form:** `<visibility> <type> <name> (<parameters>)`
 - example: `public void turnLeft() { ... }`
- The `this` keyword
 - how an instance calls a method on itself within its class definition
 - example: `this.turnRight()`

Announcements

- HW1 is out!
- Mixer tomorrow in CIT 3rd floor atrium, 5-6PM!
- Sign up on Piazza!! Link on website!
- Sections start today
 - you should have a section by now – if not, email the Head TAs ASAP (cs0150headtas@lists.brown.edu)
 - if you try to attend a section you aren't signed up for, you will not get checked off
 - find assigned room in the same link that you used to sign up
- For the best email response time: email the TA listserv!
(cs0150tas@lists.brown.edu)
 - next best: email cs0150headtas
 - slow response: email individual TA – don't do it!

