1

# CS/ENGRD 2110
# FALL 2016

Lecture 5: Local vars; Inside-out rule; constructors
http://courses.cs.cornell.edu/cs2110

## References to text and JavaSummary.pptx

2

- Local variable: variable declared in a method body
  B.10–B.11 slide 45
- Inside-out rule, bottom-up/overriding rule C.15 slide 31-32 and consequences thereof slide 45
- Use of **this** B.10 slide 23-24 and **super** C.15 slide 28, 33
- Constructors in a subclass C.9–C.10 slide 24-29
- First statement of a constructor body must be a call on another constructor —if not Java puts in **super();** C.10 slide 29

## Homework

3

Visit course website, click on Resources and then on Code Style Guidelines. Study

- 4.2 Keep methods short
- 4.3 Use statement-comments …
- 4.4 Use returns to simplify method structure
- 4.6 Declare local variables close to first use …

## Local variables

middle(8, 6, 7)

4

```
/** Return middle value of a, b, c (no ordering assumed) */
public static int middle(int a, int b, int c) {
  if (b > c) {
    int temp= b;
    b= c;
    c= temp;
  }

  if (a <= b) {
    return b;
  }

  return Math.min(a, c);
}
```

Local variable: variable declared in method body

Parameter: variable declared in () of method header

a 8 b 6 c 7

temp ?

All parameters and local variables are created when a call is executed, *before* the method body is executed. They are destroyed when method body terminates.

## Scope of local variables

5

```
/** Return middle value of a, b, c (no ordering assumed) */
public static int middle(int a, int b, int c) {
  if (b > c) {
    int temp= b;
    b= c;
    c= temp;
  }

  if (a <= b) {
    return b;
  }

  return Math.min(a, c);
}
```

block

Scope of local variable (where it can be used): from its declaration to the end of the block in which it is declared.

## Principle: declaration placement

6

```
/** Return middle value of a, b, c (no ordering assumed) */
public static int middle(int a, int b, int c) {
  int temp;
  if (b > c) {
    temp= b;
    b= c;
    c= temp;
  }

  if (a <= b) {
    return b;
  }

  return Math.min(a, c);
}
```

**Not good!** No need for reader to know about temp except when reading the then-part of the if-statement

Principle: Declare a local variable as close to its first use as possible.

## Assertions promote understanding

7

```
/** Return middle value of a, b, c (no ordering assumed) */
public static int middle(int a, int b, int c) {
  if (b > c) {
    int temp= b;
    b= c;
    c= temp;
  }
  // b <= c
  if (a <= b) {
    return b;
  }
  // a and c are both greater than b
  return Math.min(a, c);
}
```

Assertion: Asserting that b <= c at this point. Helps reader understand code below.

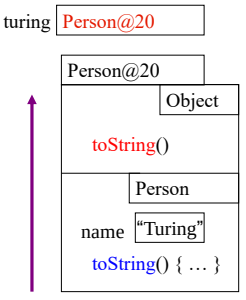
## Bottom-up/overriding rule

8

Which method toString() is called by

turing.toString() ?

**Overriding rule** or **bottom-up rule:**
To find out which is used, start at the bottom of the object and search upward until a matching one is found.




## Calling a constructor from a constructor

9

```
public class Time
  private int hr;    //hour of day, 0..23
  private int min; // minute of hour, 0..59

  /** Constructor: instance with h hours and m minutes */
  public Time(int h, int m) { hr = h; min = m; assert …; }

  /** Constructor: instance with m minutes … */
  public Time(int m) {
      hr   = m / 60;
      min = m % 60;
  }
  …
}
```

Want to change body to call first constructor

## Calling a constructor from a constructor

10

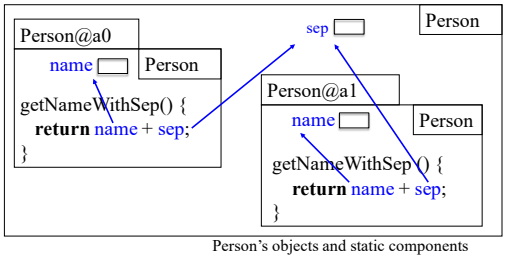
```
public class Time
  private int hr;    //hour of day, 0..23
  private int min; // minute of hour, 0..59

  /** Constructor: instance with h hours and m minutes … */
  public Time(int h, int m) { hr = h; min = m; assert …; }

  /** Constructor: instance with m minutes … */
  public Time(int m) {
     this(m / 60, m % 60);
  }
}
```

Use **this** (*not* Time) to call another constructor in the class.
Must be first statement in constructor body!

## Inside-out rule

11

***Inside-out rule:*** Code in a construct can reference names declared in that construct, as well as names that appear in enclosing constructs. (If name is declared twice, the closer one prevails.)




Person's objects and static components

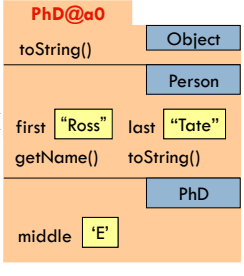
## Constructing with a Superclass

12

```
/** Constructor: person "f n" */
public Person(String f, String l) {
  first= n;
  last= l;
}

/** Constructor: PhD "Dr. f m. l"*/
public PhD(String f, char m, String l) {
  super(f, l);
  middle= m;
}

new PhD("Ross", 'E', "Tate");
```

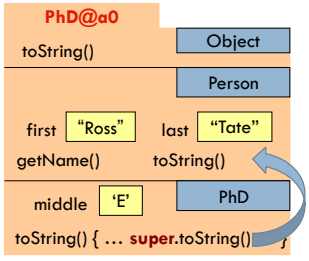
Use **super** (*not* Person) to call superclass constructor.

Must be first statement in constructor body!



## About **super**

13

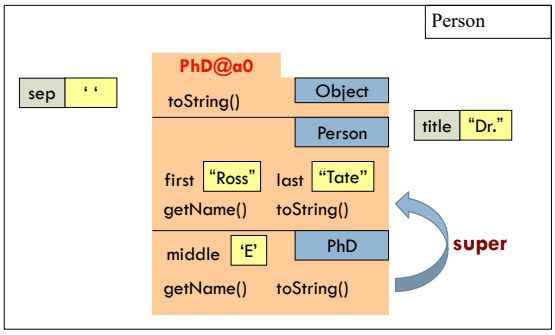



Within a subclass object, **super** refers to the partition above the one that contains **super**.

Because of keyword **super**, the call toString here refers to the Person partition.

## Bottom-Up and Inside-Out

14




## Without OO …

15

Without OO, you would write a long involved method:

```
public double getName(Person p) {
  if (p is a PhD)
    { … }
  else if (p hates formality)
    { … }
  else if (p prefers anonymity)
    { … }
  else …
}
```

OO eliminates need for many of these long, convoluted methods, which are hard to maintain.

Instead, each subclass has its own getName.

Results in many overriding method implementations, each of which is usually very short