References to text and JavaSummary.pptx

- 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

1. 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 …

Scope of local variable

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

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

Principle about placement of declaration

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

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**

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

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

**Bottom-up/overriding rule**

Which method `toString()` is called by:

```java
Butterfly@20
```

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.

**Inside-out rule**

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

```java
Person@a0
n
getNAndPop() {
    return n + PersonPop;
}
```

**Parameters participate in inside-out rule**

Doesn’t work right

```java
Person@a0
n
setN(String name) {
    n = name;
}
```

Parameter `n` "blocks" reference to field `n`.

**A solution: use this**

Memorize: Within an object, `this` evaluates to the name of the object.

```java
Person@a0
n
setN(String name) {
    this.name = n;
}
```

In object `Person@a0`, `this` evaluates to `Person@a0`.

In object `Person@a1`, `this` evaluates to `Person@a1`.

**About super**

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

```java
Object
toString()
```

```java
Butterfly
toString() {
    Object
    toString();
    Butterfly
    toString() { … }
}
```

Because of the keyword `super`, this calls `toString` in the `Object` partition.
Calling a constructor from a constructor

```java
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) { ... }
/** Constructor: instance with m minutes */
public Time(int m) {
    hr = m / 60;
    min = m % 60;
} ... Want to change body to call first constructor
```
**Principle: initialize superclass fields first**

```java
/** Constructor: employee with name n, year hired d, salary s */
public Employee(String n, int d, double s) {
    // Employee constructor
}

/** Constructor: executive with name n, year hired d, salary of $50,000, bonus b */
public Executive(String n, int d, double s, double b) {
    Employee(n, d, 50000);
    bonus = b;
}
```

To call a superclass constructor, use `super( ... )`.

---

**Java syntax:** First statement of any constructor you write must be a call on another constructor `this( ... )` or `super( ... )`.

If you don’t put one in, Java inserts this one: `super();`