1

CS/ENGRD 2110 SPRING 2019

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

Announcements

2

- A1 is due Thursday
  If you are working with a partner: form a group **well before submitting** on CMS. After forming group, **only one** you needs to submit.
- This week's recitation is on testing. No tutorial/quiz this week.
- A2 is out now

Today's Topics

3

- Scope, local variables, inside-out rule
- Overloading, bottom-up rule
- Constructors, this, default, super

All searchable in JHT!

JavaHyperText
Filter: constructor
HERE is a summary of all important points concerning constructors: pdf file.

Local Variables

4

middle(8, 6, 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;
  }

  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 limited in **scope**…

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.

Scope In General: Inside-out rule

6

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.)

```
public class C {
  private int field;
  public void method(int parameter) {
    if (field > parameter) {
      int temp= parameter;
    }
  }
}
```

block
method
class

What 3 numbers are printed?

7

```
public class ScopeQuiz {
  private int a;
  public ScopeQuiz(int b) {
    System.out.println(a);
    int a= b + 1;
    this.a= a;
    System.out.println(a);
    a= a + 1;
  }
  public static void main(String[] args) {
    int a= 5;
    ScopeQuiz s= new ScopeQuiz(a);
    System.out.println(s.a);
  }
}
```

A: 5, 6, 6
B: 0, 6, 6
C: 6, 6, 6
D: 0, 6, 0

Review: Constructor

8

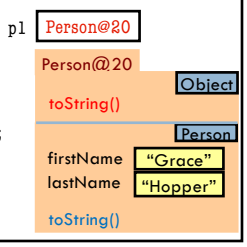
```
public class Person {
  private String firstName; // not null
  private String lastName;

  /** Constructor: a Person with first and last names f, l.
    * Precondition: f is not null. */
  public Person(String f, String l) {
    assert f != null;
    firstName= f;
    lastName= l;
  }
}
```

Constructor:
Initialize fields to truthify class invariant

Review: Which toString is called?

9

```
public class Person {
  private String firstName;
  private String lastName;

  public Person(String f, String l) {
    assert f != null;
    firstName= f; lastName= l;
  }
  public String toString() {
    return firstName + " " + lastName;
  }
}
Person p1= new Person("Grace", "Hopper");
p1.toString();
```

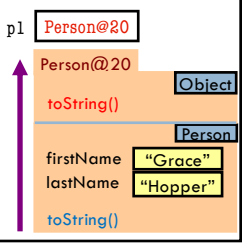


Which toString : Bottom-up Rule

10

```
public class Person {
  private String firstName;
  private String lastName;

  public Person(String f, String l) {
    assert f != null;
    firstName= f; lastName= l;
  }
  public String toString() {
    return firstName + " " + lastName;
  }
}
Person p1= new Person("Grace", "Hopper");
p1.toString();
```

Which method is used? Start at bottom of the object and search upward until you find a match.



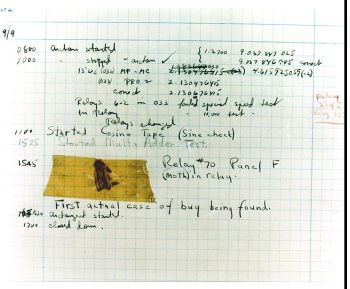
Grace Hopper

11

(1906-1992)
Rear Admiral, US Navy
PhD, Math, Yale, 1934
Pioneering computer programmer
Posthumous Presidential Medal of Freedom (highest civilian honor), 2016

Grace Hopper: "bug"

12



Middle Names (v1)

13

```
public class Person {
   private String firstName; //not null
   private String middleName;
   private String lastName;
   /** Constructor: ... */
   public Person(String f, String l)
      { assert f != null; firstName= f; lastName= l; }
   /** Constructor: ... */
   public Person(String f, String m, String l) {
      assert f != null;
      firstName= f;
      middleName= m;
      lastName= l;
   }
}
```

👍 or 👎 ?

Better: reuse first constructor without copying code

Middle Names (v2)

14

```
public class Person {
   private String firstName; //not null
   private String middleName;
   private String lastName;
   public Person(String f, String l) {
      assert f != null;
      firstName= f;
      lastName= l;
   }
   public Person(String f, String m, String l) {
      this(f, l);
      middleName= m;
   }
}
```

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

Default Constructor

15

```
public class Person {
   private String firstName; //not null
   private String lastName;

   public Person() { };
}

Person p= new Person();
```

👍 or 👎 ?

Default Constructor

16

```
public class Person {
   private String firstName; //not null
   private String middleName;
   private String lastName;
   public Person(String f, String l) {
      assert f != null;
      firstName= f;
      lastName= l;
   }
   public Person(String f, String m,
      this(f, l);
      middleName= m;
   }
}

Person p= new Person();
```

Nope!

Syntax Error: No constructor in Person matches this invocation
Arguments: ()
Expected return type: Person
Candidate signatures:
Person(String, String)
Person(String, String, String)

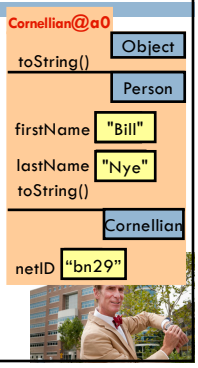
Super Constructor

17

```
public class Cornellian extends Person {
  private String netID;

  /** Constructor: Person with a netID. */
  public Cornellian(String f, String l, String id) {
    super(f, l);
    netID= id;
  }

}

new Cornellian("Bill", "Nye", "bn29");
```

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

Must be first statement in constructor body!



Super Constructor

18

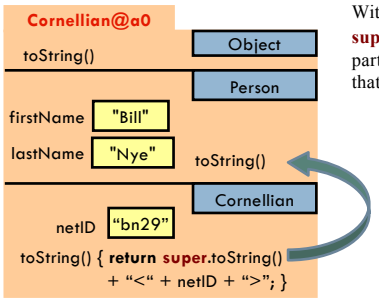
```
public class Cornellian extends Person {
  private String netID;

  /** Constructor: Person with a netID. */
  public Cornellian(String f, String l, String id) {
    super();
    netID= id;
  }

}
```

If first statement in constructor body is not a constructor call, Java inserts super(); for you!

👍 or 👎 ?

More about **super**

19



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.

Grace Hopper

20