Example Constructor: Person Class

```java
public class Person {
    private String firstName; //cannot be null
    private String lastName;

    /** Create a person with the given names. */
    public Person(String f, String l) {
        assert f != null;
        firstName = f;
        lastName = l;
    }
}
```

Adding a Middle Name Option (v1)

```java
public class Person {
    private String firstName; //cannot be 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) {
        assert f != null;
        firstName = f;
        middleName = m;
        lastName = l;
    }
}
```

Adding a Middle Name Option (v2)

```java
public class Person {
    private String firstName; //cannot be 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;
    }
}
```

Too Busy for Constructors? Java makes one!

```java
public class Person {
    private String firstName; //cannot be null
    private String lastName;

    public Person() {}

    public String toString() {
        return firstName + " " + lastName;
    }
}
```

Announcements

- A1 is due tomorrow
  - If you are working with a partner: form a group on CMS & submit once

- A2 is out. Remember to get started early!
  - Because of Jewish holidays, the due date for A2 has been changed to Sunday, 16 September.
  - ONLY TWO DAYS OF LATENESS ALLOWED.
    - Last day to submit is 18 September.

- Next week’s recitation is on testing. No tutorial/quiz this week!
Wow, every class has an empty Constructor?

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

Person p= new Person();

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

Nope!

```java
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();

Constructing with a Superclass

```java
public class Cornellian extends Person {
    private String netID;
    /** Constructor: Person with a netID. */
    public Cornellian(String f, String l, String id) {
        super(f, l); //call superclass constructor.
        netID = id;
    }
}
```

new Cornellian("David", "Gries", "djg17");

Not Feeling Super? Java thinks you are!

```java
public class Cornellian extends Person {
    private String netID;
    /** Constructor: Person with a netID. */
    public Cornellian(String f, String l, String id) {
        super(); //if first statement in constructor body
        if first statement in constructor body
        is not a constructor call, Java inserts
        super(); for you!
        netID = id;
    }
}
```

new Cornellian("David", "Gries", "djg17");

More about super

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

```java
public class Cornellian extends Person {
    private String netID;
    /** Constructor: Person with a netID. */
    public Cornellian(String f, String l, String id) {
        super(); //call superclass constructor.
        netID = id;
    }
    public String toString() {
        return super.toString() + " " + netID;
    }
}
```

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

Without OO ...

```java
public double getName(Person p) {
    if (p is a CornellUndergrad) {
        ... }
    else if (p is a CornellFaculty) {
        ... }
    else if (p prefers Cornell) {
        ... }
    else ...
}```
/** Return middle value of a, b, c (no ordering assumed) */
public static int middle(int a, int b, int c) {
    int temp = b;
    b = c;
    c = temp;
    if (a <= b) {
        return b;
    }
    return Math.min(a, c);
}

Scope of local variables

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

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.)

/** A useless class to illustrate scopes*/
public class C {
    private int field;
    public void method(int parameter) {
        int temp = parameter;
        if (field > parameter) {
            temp = parameter;
        }
        if (a <= b) { return b; }
        return Math.min(a, c);
    }
}

Principle: declaration placement

/** 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);
    // Principle: Declare a local variable as close to its first use as possible.
}

Poll time! What 3 numbers are printed?

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, 6, 6};
        ScopeQuiz s = new ScopeQuiz(a);
        System.out.println(a); // A: 5, 6, 6
    }
}