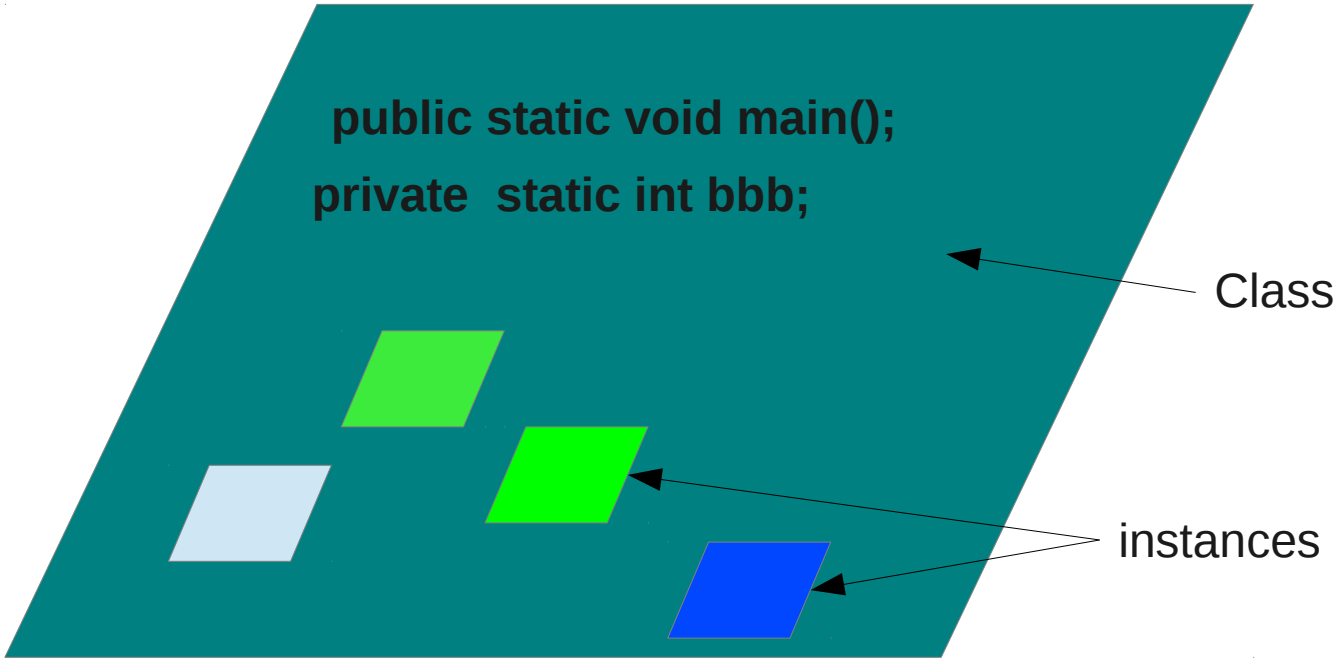
Lecture 10
Declarations and Scope

Declarations and Scope

- We have seen numerous “qualifiers” when defining methods and variables
  - `public`
  - `private`
  - `static`
  - `final`
  - (we'll talk about `protected` when formally addressing inheritance)

Static

- Static means “associated and stored” with the class
  - Can modify a method declaration
  - Can modify a class variable declaration



Some important properties of static

- An instance does NOT have to first be constructed before invoking a static method or accessing a static variable
  - The storage for static variables and methods is defined when the class is compiled with javac
  - This is also why the main method must be declared static
    - No instance of the class has been constructed before main() is invoked
    - At run time "`$ java MyClass`" is making the following method call
      - `MyClass.main(args)`

What can be called where

```
public class MyClass {
   public static void aMethod(){};
   public void bMethod();
   public static int aVariable;
   public int bVariable;
}
```

```
{
   MyClass anInstance = new MyClass();
}
```

Referring to Class	Referring to an Instance
MyClass.aMethod();	anInstance.aMethod();
MyClass.aVariable;	anInstance.aVariable;
~~MyClass.bMethod();~~	anInstance.bMethod();
~~MyClass.bVariable;~~	anInstance.bVariable;

You can reference class variables and methods (those declared static) from an instance.

You cannot reference instance variables or methods (those not declared static) without first constructing an instance

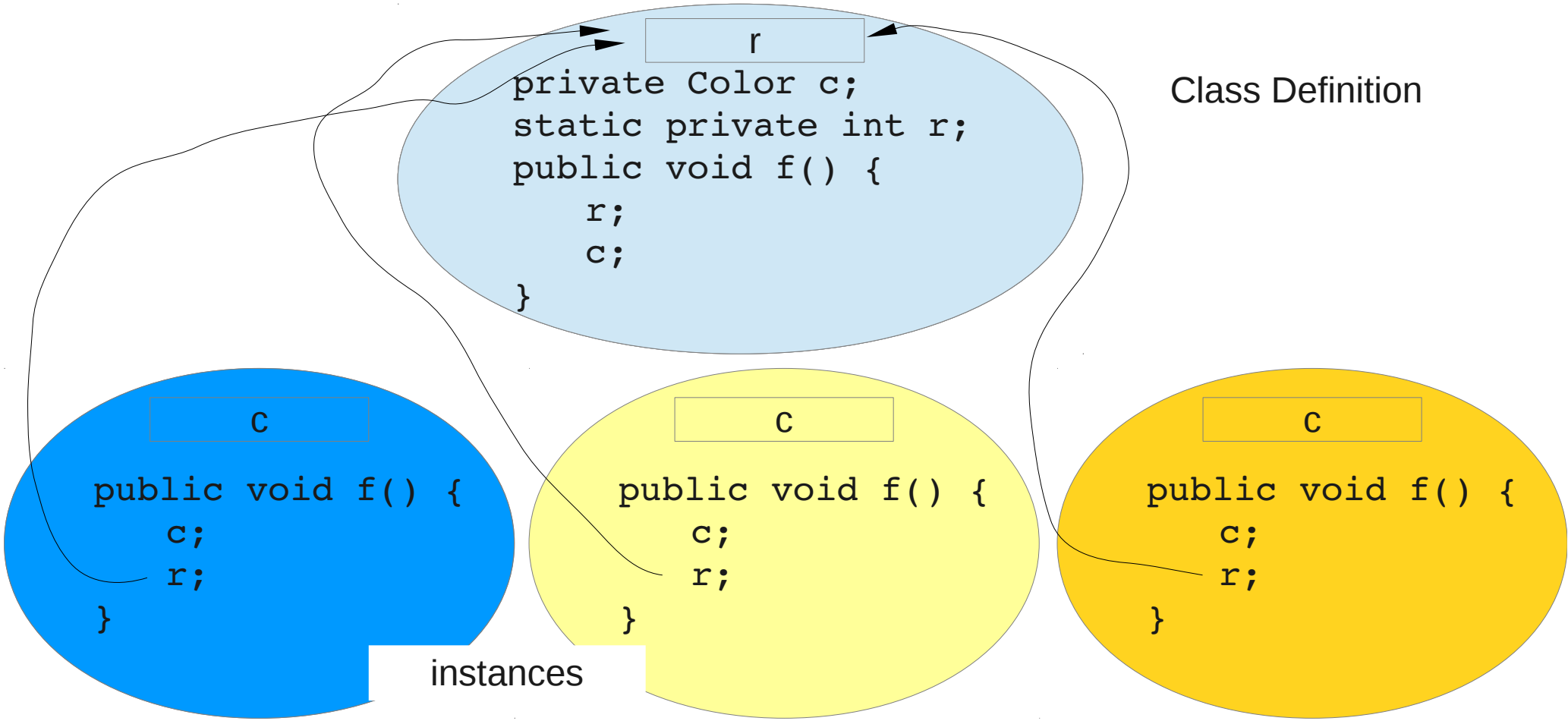
Variable Initialization

- Java initializes class (static) and instance variables to “zero”
  - `static double x;   // x == 0.0`
  - `static char c;     //  c has ascii value 0`
  - `static FilledOval f; // f references a null instance`
  - `private int i;  // i = 0 when an instance is`
`//  --> constructed <--`
  - `static boolean b;  // b has initial value of False`

- Temporary variables are <u>never</u> initialized
  - Good (defensive) coding never assumes that any variable is initialized. Program should explicitly do this step
    - Why? Other languages (e.g., C) do no default initialization.

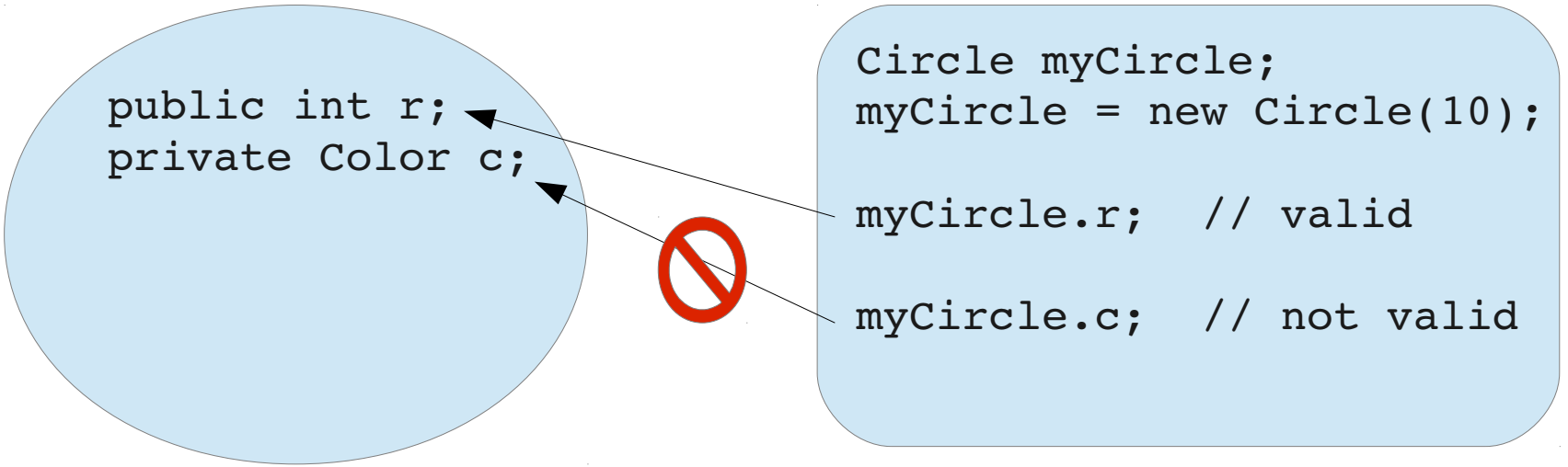
Another view of the static modifier

- Static variables are stored with the class, the value in the static variable is shared by all instances
- Non-static variables are stored with the instance



Public vs. Private (access modifiers)

- Public Variables and Methods can be seen/called by all *other* classes



- `private` means that only instances of the class have access to the variable
- If class B extends class A, B does NOT have access to A's private variables

final

- final means “cannot be changed”
  - We declare constants to be final
    - Constants must also be declared static.
      - Why? Because constants must be defined without any constructed instances. That can only occur with the static modifier
      - Math.E, Math.PI are static final. One does not need a Math instance to reference E and PI.
      - We conclude that Math.abs() must also be declared static since we can call it without first constructing a Math instance
      - http://docs.oracle.com/javase/6/docs/api/java/lang/Math.html
  - (Later, when describing inheritance, we'll learn about declaring methods and classes to be final and what that means)

Temp Variables and Statement Blocks

```
{                                                   // block A Start
   int i = 50;
   int k =100;
   {                                                // block B  Start
      int j;
   }                                                // block B  End
   j=10;   // <-- j is NOT defined
}                                                   // block A  End
```

- It is legal to define a temporary variable inside *any* statement block.
- Block B is contained in Block A (or B is nested in A).
  - A variable definition is relative to that block and available to any blocks that is encloses
  - i and k are available in Blocks A&B. j is only available to block B.

Scope of variables

- Scope defines when a variable is available.
- If myMethod() below is invoked, what does it return?

```
public int myMethod()
{
   int retval, itemp=100;
   retval = itemp;
   {
      int retval, itemp = 75;
      retval = itemp;
   }
   return retval;
}
```

ANS: 100

Scope of variables

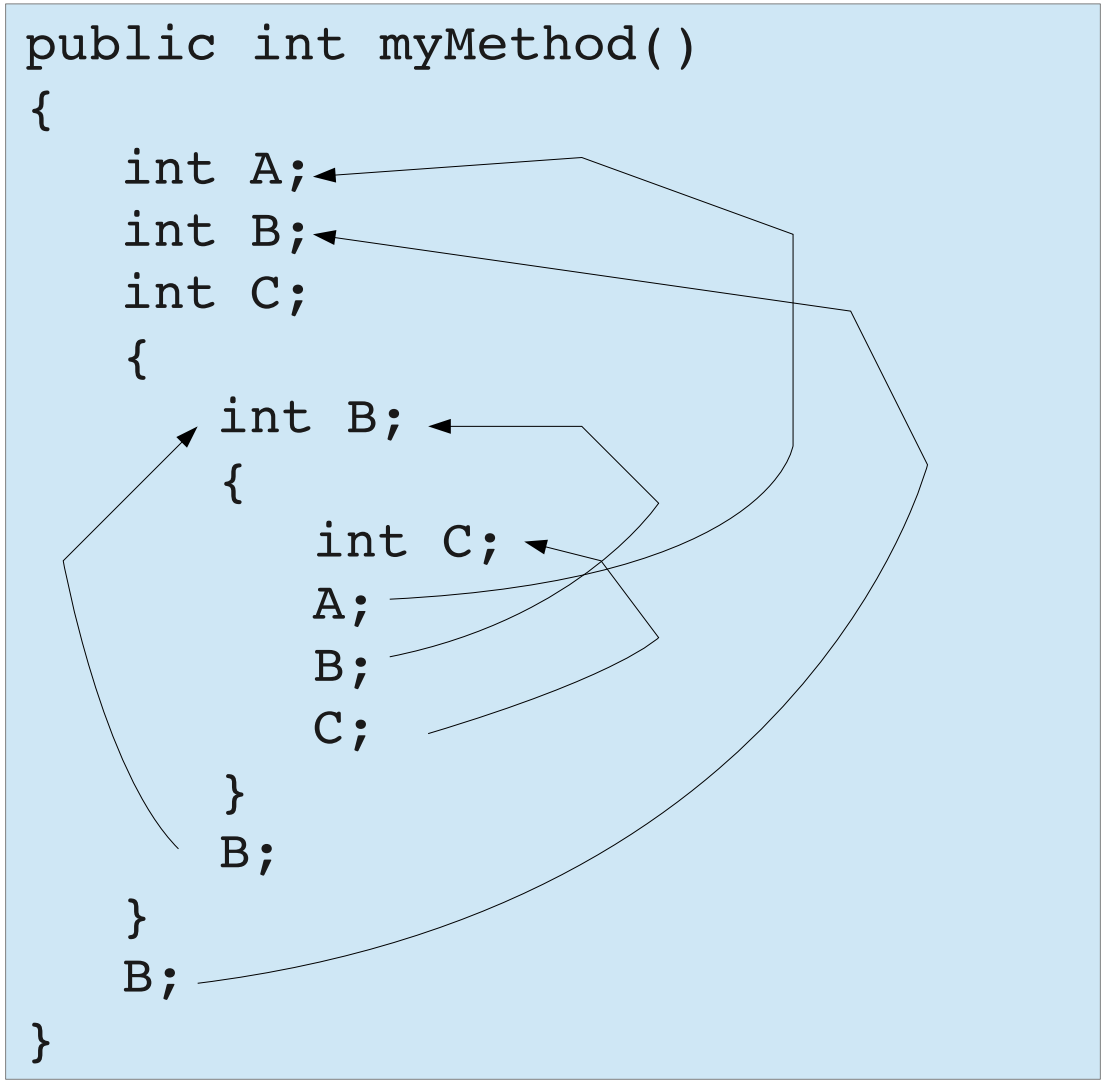
- How java figures out which version of an identifier is accessed (this is the scope)
- It looks into the current statement block, if the variable is defined, it uses it. Otherwise it looks into the next enclosing block. It keeps looking at enclosing scopes (blocks)

```
public int myMethod()
{
   int retval, itemp=100;
   retval = itemp;
   {
      int temp = 75;
      retval = itemp;
   }
   return retval;
}
```

What is scope

- When you have the same identifier name declared in multiple statement blocks, Java has to figure out which one of these identifier's storage should be used
- It gets the version of the identifier defined “closest” to where it is used.
- Inside a statement block is first (and any enclosing blocks),
- Then inside the method (temporary vars)
- Then inside the instance
- Then inside the class
- If it cannot find it in any of these “scopes”, the identifier is not defined.

Which Identifier is “in Scope”?



- B and C are defined in multiple blocks
- When referenced in a block, Java must look up which version of the identifier is in scope

What about instance variables. Where are they available?

```
public class myClass()
{
   private int A;
   private int B;
   private int C;
   public MyClass (){
      int B;
      {
         int C;
         A;
         B;
         C;
      }
      B;
   }
 }
```

- Instance variables are available to all methods and constructors defined in the class
- Same logic as the previous slide.
- Just look at the enclosing statement blocks
- Note B and C inside constructor are Shadow variables. This is bad.

Why bad naming is confusing

- DO NOT name temporary variables the same as instance or class variables
  - Legal Java but very confusing
  - This is called a shadow variable (and is usually a very bad idea)

```
public class Scoper{
{
   private int state;
   public double scopedMethod()
   {
      double state = 99.9;    // Shadows the instance var
      return state;
   }
   public int scoop()
   {
      return state;         // this is the instance variable
   }
}
```

Revisit `this`

- This is read/translated as “this instance of the class”
- You can use it access a the instance's version of a shadowed variable

```
public class Shadow{
{
   private String address;
   public Shadow(String address)
   {
      this.address = address;
   }
}
```

When to use instance versus temporary variables

- Instance variables
  - Data stored in the variable is needed by multiple methods of the class
    - e.g. mousePressed boolean in determining when to drag an image
  - Data stored in the variable is needed across multiple invocations of the same method
    - e.g. lastPoint in many of the onMouseDrag() methods
- Temporary variables
  - “If you can, make a variable temporary!”
  - Scratchpad storage, needed only for the duration of a method call, and then can tossed away.

Coding Style Guide

- Why do we need coding style guidelines?
  - A great deal of time as a programmer is spent reading other people's code
    - Code review
    - Want to understand how an algorithm is implemented
    - Need to debug code you didn't write
    - Want to “steal” (borrow, copy) a subsection of code and incorporate into your own
    - Want to start from something existing and modify
  - Adhering to a coding style makes is easier to understand your own code

Coding Style Guide

- Indentation
  - Tabs vs. Spaces (I recommend tabs)
  - Where to put '{' and '}' for statement blocks
- Comments
  - Informative, not redundant
    - `i=j; // Assign i the value of j` <<- not a good comments, adds nothing but clutter
  - Not too many, not too sparse. Comment blocks of code
- Constant, variable, method, class naming
- See Style Guide
  -