CSE 142

Declarations and Scope

Outline for Today

- Goal: present more precisely several things we've dealt with informally up to now
  - Only key topics at this time; won't cover all the technical details
- Scope defined
- Scope for instance variables and methods
  - Public and private
  - Using local methods
  - Accessing instance variables in other objects
  - "this"
- Scope for method parameters and local variables

Declarations

- Everything in a Java program is referenced using an identifier (name)
- New names must be ***declared***
  - Class declarations
  - Method definitions and instance variable declarations in a class
  - Parameter and local variable definitions in methods

Scope

- The *scope* of a declaration is the region of the program where that declaration is in effect
  - Classes: other classes in the program
  - Methods and instance variables: the class containing the declaration and, possibly, other classes if they are public
  - Parameters and local variables: part or all of the body of the method containing the declaration
- Scope limits the range of a declaration
  - Allows sensible reuse of names (identifiers) in different parts of the code

Methods and Instance Variables – Class Scope

- These are declared inside a class
- Scope depends on whether declared ***public*** or ***private***
  - Always accessible inside the class
  - Accessible to clients outside the class if declared public
  - Not accessible to clients if declared private
- Inside the class, local methods and instance variables can be referenced by their simple names
- Always use public or private in CSE142
  - There are rules about what happens if you leave these off; we'll simplify our life by not dealing with them

Example – HuskyCard Class

```
public class HuskyCard {
  private int balance;   // card balance
  private String name; // cardholder

  /** Deposit money ... */
  public boolean deposit(int amount) {
     balance = balance + amount;
   }

  /** Report current balance ... */
  public int getBalance ( ){
     ...
  }
```

- Identifiers getBalance and deposit are visible inside and outside class HuskyCard
- Identifiers balance and name are only visible inside the class

Parameters – Method Scope

- The scope of a parameter declaration is the body of the method or constructor containing the parameter declaration

```
/** deposit amount in this HuskyCard */
public void deposit(double amount) {
     ...
}
/** Construct new HuskyCard ... */
public HuskyCard(String ownerName, int idNumber) {
     ...
}
```

- When the method is called, each parameter is initialized by assigning it the corresponding argument value in the method call

```
HuskyCard card = new HuskyCard("B. Moose", 6834654);
card.deposit(4217);
```

Example – HuskyCard Class

```
public class HuskyCard {
  private int balance;   // card balance
  private String name; // cardholder

  /** Deposit money ... */
  public boolean deposit(int amount) {
    balance = balance + amount;
   }

  /** Report current balance ... */
  public int getBalance ( ){
     ...
  }
```

- Identifier amount is only visible inside the deposit method
- Identifier balance is visible to all methods in this class

Local Variables

- Local variables can be declared inside a method
  - Provides scratch space for temporary values and intermediate calculations
  - Scope extends to the right brace "}" matching the nearest preceding left brace "{"

    This can hide a instance variable, parameter, or local variable declared in a surrounding scope – generally bad style; don't do it
  - Variable no longer exists after leaving the scope

    (in particular, parameters and local variables no longer exists after method execution ends)

Example

- Suppose we're programming a payroll system and want to calculate employee pay.

```
/** return the weekly pay of this Employee */
public double getWeeklyPay( ) {
    double basePay;
    double overtimePay;
    if (hours <= 40) {
      basePay = hours * rate;                // hours, rate are instance variables
      overtimePay = 0.0;
    } else {
      basePay = 40 * rate;
      overtimePay = 1.5 * (hours-40) * rate;
    }
    return basePay + overtimePay;
}
```

Trace

```
Employee intern = new Employee(…);
System.out.println(intern.getWeeklyPay());
```

Variable Declaration with Initialization

- A variable declaration can also specify an initial value

```
/** Return the area of the circle with given diameter */
public double area(double diameter ) {
    double radius = diameter / 2.0;
    return 3.14 * radius * radius;
}
```

- Common for temporary quantities used inside a method
  - Can make code easier to read if you name intermediate results by declaring and initializing appropriate local variables
- Less common for instance variables
  - Usually better style to put all initializations in the constructor(s)

Nested Scopes

- **The scope of a parameter declared in a method is nested inside the class scope containing instance variables and methods belonging to the class**
- **The diagrams we use for a method call are designed to show this explicitly**

  (The book uses a slightly different diagramming convention, but it's easy to move back and forth)
- **If a name is referenced in a method, to find the actual thing the name refers to:**
  - **First check the method scope**
  - **Then, if you don't find it, look at the surrounding class (object) scope**
  - **If still not found, it is not declared – compiler will complain**

Nested Scopes Diagramed

- **Example**

```
HuskyCard card = new HuskyCard("B. Moose", 6834654);
card.deposit(4217);
```

Nested Scope Pitfall

- **Some (buggy) code**

```
public class HuskyCard {
   private String name;          // cardholder's name
   /** Change the name on this card */
   public void setName(String name) {
      name = name;
   }
}
```

- **What happens if we execute this?**

```
HuskyCard card = new HuskyCard("B. Moose", 6834654);
card.setName("R. Squirrel");
```

Draw the Diagram

- **Is there a way to…**

  Yes, there is a way to reference the instance variable even though the parameter has the same name; stay tuned…

Scopes Revisited

- **Another version of pay calculation. What happens here?**

```
/** return the weekly pay of this Employee */
public double getWeeklyPay( ) {
    if (hours <= 40) {
      double basePay = hours * rate;
      double overtimePay = 0.0;
    } else {
      double basePay = 40 * rate;
      double overtimePay = 1.5 * (hours-40) * rate;
    }
    return basePay + overtimePay;
}
```

  - **(Hint: what is the scope of a local variable declaration?)**

Another Scope Glitch

- **Consider the following code**

```
sum = 0;
for (int n = 1; n <= 100; n++) {
    sum = sum + n;
}
System.out.println("final value of sum is " + sum + " and final value of n is " + n);
```

  - **This isn't legal (and won't compile). Why not?**

Scopes and Multiple Objects

- **Each object defines a separate scope for its instance variables and methods**
- **A method or instance variable in another object can be accessed if it is public or declared in the same class by writing**

  objectName . methodName (…);

  - **or**

  objectName . instanceVariableName
- **When a method executes, think of its local scope as being surrounded by the scope of the corresponding object**

Example: HuskyCard Transfer

```
class HuskyCard {
    …
    /** Transfer given amount from HuskyCard */
    public void transferFrom(double amount, HuskyCard otherCard) {
        boolean success = otherCard.withdraw(amount);
        if (success) {
            balance = balance + amount;
        }
    }
```

Execution Example

```
HuskyCard yours = new HuskyCard (“Chris”, 567);
yours.deposit(5000);
HuskyCard mine = new HuskyCard(“Me”, 1234);
mine.transferFrom(yours, 2000);
```

Another Implementation of Transfer

```
class HuskyCard{
   …
   /** Transfer given amount from otherCard */
   public void transferFrom(double amount, HuskyCard otherCard) {
      if (otherCard.balance >= amount) {
         otherCard.balance = otherCard.balance – amount;
         balance = balance + amount;
      }
   }
```

- **Discuss: Is this better or worse than using otherCard.withdraw(...)? Why or why not?**

Method and Instance Variable Names, Revisited

- **When we write something like**

  `name = studentName;`

  - **or**

  `otherCard.balance = otherCard.balance – amount;`

  the simple occurrence of “name” or “balance” refers to fields in the current object scope where the method is executing
- **But technically, every method or instance variable has a full name, which is always *objectName . fieldName*.**
- **When we use a simple name like balance by itself, we really mean *“the current object” .* balance**

“The Current Object” – this

- **Java has a reserved keyword, *this*, that can be used to explicitly refer to “the current object”**
- **If we use a field name by itself**

  `balance = 42.17;`

  it is equivalent to writing

  `this.balance = 42.17;`
- **You can write this explicitly if you want. If you don’t, Java interprets the simple *name* as meaning this.*name***

“this” as an Implicit Parameter

- When we send a message to an object (call a method)

```
card.deposit(1000)
```

  “this” is automatically provided and initialized to refer to the object receiving the message
- In effect, it is an invisible parameter

“this” – What Really Happens

- What you write:

```
public class HuskyCard {
    int balance;
    /** deposit amount in … */
    public void deposit(int amount){

        balance = balance + amount;

    }
}

card.deposit(100);
```

- What Java does for you behind the scenes

```
public class HuskyCard {
  int balance;
  /** deposit amount in … */
  public void deposit(HuskyCard this,
                      int amount){
     this.balance =
            this.balance + amount;
  }
}

deposit(card, 100);
```

- (This is quite technical, but you should understand the general idea)

A Common Use for this

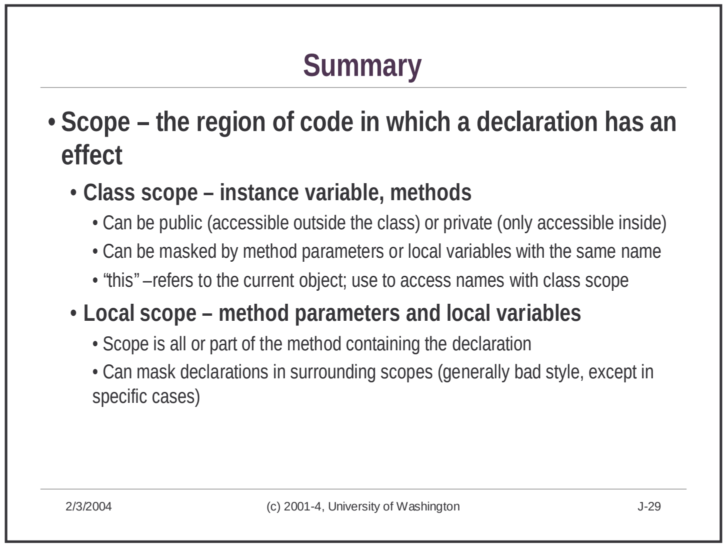
- Normally instance variables and local variables or parameters should not have the same name for style and readability
- But in constructors and sometimes in methods, it’s often more readable if parameter names match the fields they initialize
  - If you’ve picked a good name for one it’s often the best name for the other
- Use “this” to access an instance variable whose scope is masked by a local parameter declaration

```
/** Construct new HuskyCard … */
public HuskyCard(String name, int number) {
    this.name = name;
    this.number = number;
}
```

Scope Rules and This

- Trace execution of

```
HuskyCard test = new HuskyCard(“scope demo”, 654);
```


Summary

- Scope – the region of code in which a declaration has an effect
  - Class scope – instance variable, methods
    - Can be public (accessible outside the class) or private (only accessible inside)
    - Can be masked by method parameters or local variables with the same name
    - "this" –refers to the current object; use to access names with class scope
  - Local scope – method parameters and local variables
    - Scope is all or part of the method containing the declaration
    - Can mask declarations in surrounding scopes (generally bad style, except in specific cases)

2/3/2004 (c) 2001-4, University of Washington J-29