CSE 142

Declarations and Scope

Outline for Today

- Goal: present more precisely several things we've dealt with informally up to now
  - Only essential topics for CSE142; 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
  - 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

- 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 – Tile Class

```
public class Tile {
  private int size;   // tile size

  /** add picture of this tile... */
  public void addTo(...) {
    ...
    display(...);
  }

  // draw a tile at the right place
  private void display(..., Shape s, ...) {
    s.moveBy(...size...);
    ...
  }
```

- Identifiers Tile and addTo are visible inside and outside class Tile
- Identifiers size and display are only visible inside the class

Parameters

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

```
/** deposit amount in this BankAccount */
public void deposit(double amount) {
    …
}
/** Construct new BankAccount with given name and account number */
public BankAccount(int accountNumber, String accountName,) {
    …
}
```

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

```
BankAccount savings = new BankAccount(12, "D. Warbucks");
savings.deposit(42.17);
```

Nested Scopes

- The scope of a parameter declaration is nested inside the scope of instance variables and methods belonging to the class
- The diagrams we use for a method call are designed to show this explicitly
- If a name is referenced in a method, to find the actual thing it 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

```
BankAccount savings = new BankAccount(567, "Rainy Day");
savings.deposit(100.00);
```

Nested Scope Pitfall

- Some (buggy) code

```
public class BankAccount {
    private String name;          // name on the account
    /** set the account name */
    public void setName(String name) {
        name = name;
    }
}
```

- What happens if we execute

```
BankAccount credit = new BankAccount(567, "Funny Money");
credit.setName("plastic");
```

Draw the Diagram

Local Variables

- Local variables can be declared inside a method
  - Provides scratch space for temporary values
  - Scope extends to the right brace "}" matching the nearest preceeding 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

```
/** 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
- Not common for instance variables
  - Better style is to put all initializations inside the constructor(s)

Scopes and Initialization

- 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?)

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 in the same class) by writing

  objectName . methodName (…);
  - or

  objectName . variableName
- When a method executes, its local scope is surrounded by the scope of the corresponding object

Example: BankAccount Transfer

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

Execution Example

```
BankAccount yours = new BankAccount(567, "Moneybags");
yours.deposit(5000.00);
BankAccount mine = new BankAccount(1234, "Me");
mine.transferFrom(yours, 2000.00);
```

Another Implementation of Transfer

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

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

Method and Instance Variable Names, Revisited

- **When we write something like**
  `name = otherAccount.name;`
  - **or**
  `otherAccount.balance = balance;`

  the 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 understands that that is what you mean**

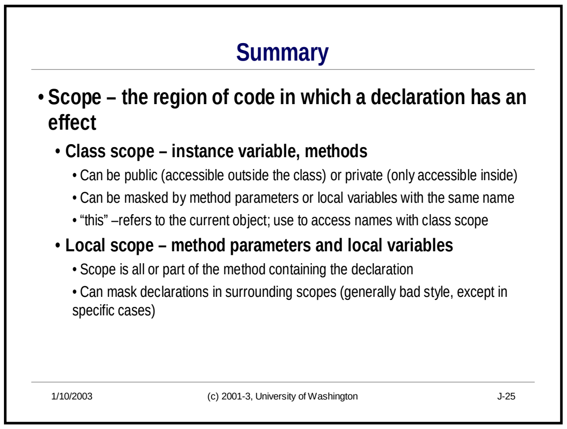
A Common Use for this

- **Normally instance variables and local variables or parameters should not have the same name**
  (Style/readability issue)
- **But in constructors, it's often more readable if parameter names are the same as the fields they initialize**
- **Use "this" to access an instance variable whose scope is masked by a local parameter declaration**

```
/** construct a new BankAccount with the given name and number */
public BankAccount(int number, String name) {
    this.number = number;
    this.name = name;
}
```

Scope Rules and This

- **Trace execution of**
  `BankAccount test = new BankAccount(654, "scope demo");`



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)

1/10/2003 (c) 2001-3, University of Washington J-25