



Controlling Access to a Class

- public, private, protected (default)



Controlling Access to a Class

- **Specifier class subclass package world**
- **private X**
- **protected X X**
- **public X X X X**
- **package X X**



Controlling Access to a Class

- Make the member public. Then everybody, everywhere, can access it
- Private :you can't touch that.
The private keyword that means no one can access that member except that particular class. Other classes in the same package cannot access **private member**
- Package/Friendly:Same Package classes can touch that
- **protected:Cross package inheritance**



Work with the **final** keyword

- The final keyword has slightly different meanings depending on the context, but in general it says “ This cannot be changed.”
The final can be used: **for data, methods and for a class**



Important notes to access a class

- There can be only one **public** class per file. It can have as many supporting “friendly” classes as you want.
- The name of the public class must **exactly match** the name of the file.
- The class cannot be **private or protected**



Important notes to access a class

- **Variable:** An item of data named by an identifier. Each variable has a type, such as int or Object, and a scope. See also *class variable*, *instance variable*, *local variable*.



Important notes to access a class

- ***instance variable***

Any item of data that is associated with a particular object. Each instance of a class has its own copy of the instance variables defined in the class. Also called a *field*.



Important notes to access a class

- Static key word:
 - ✍ Class method
 - ✍ Class variable



Important notes to access a class

- *class variable*

A data item associated with a particular class as a whole--not with particular instances of the class. Class variables are defined in class definitions. Also called a *static field*.



Important notes to access a class

- ***method***

A function defined in a class. There are *instance method*, *class method*. Unless specified otherwise, a method is not static.

instance method

Any method that is invoked with respect to an instance of a class.



Important notes to access a class

class method

A method that is invoked without reference to a particular object. Class methods affect the class as a whole, not a particular instance of the class. Also called a *static method*.



Understanding Instance and Class Members

- When you declare a member variable such as aFloat in MyClass:

```
class Test
{
    int i=47;
}
```

you declare an *instance variable*.



Understanding Instance and Class Members

```
class StaticTest
{
    static int i=47;
}
```



Understanding Instance and Class Members

```
public class TestDriver {
    public static void main(String[] args)
    {
        Test ts1=new Test();
        Test ts2=new Test();
        ts1.i=ts1.i+1;
        System.out.println("object ts1 i
        =" +ts1.i);
        System.out.println("object ts2
        i=" +ts2.i);
    }
}
```



Understanding Instance and Class Members

- What is the output? Why?



Understanding Instance and Class Members

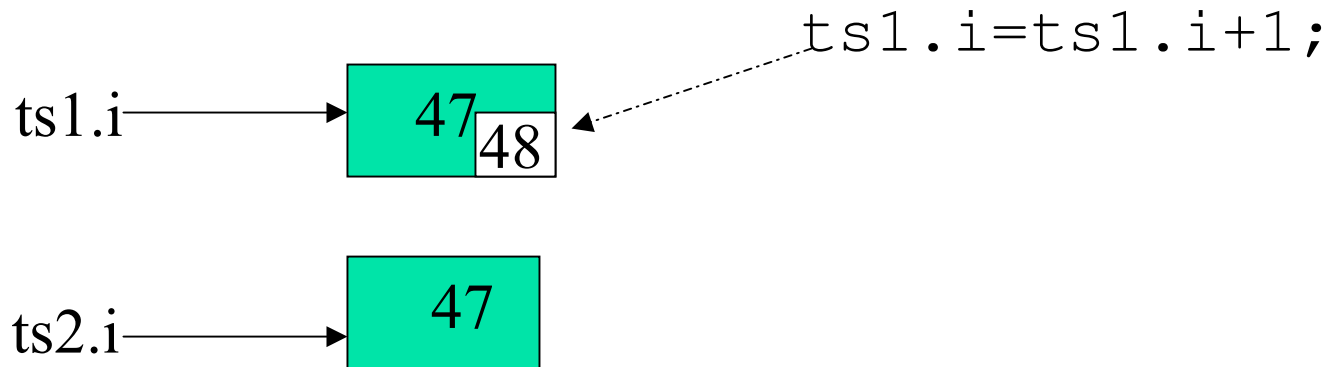
- `public class StDriver {`
- `public static void main(String[] args) {`
- `StaticTest.i=StaticTest.i+1;`
- `StaticTest ts1=new StaticTest();`
- `StaticTest ts2=new StaticTest();`
- `ts1.i=ts1.i+1;`
- `System.out.println("object ts i`
`="+StaticTest.i);`
- `System.out.println("object ts1 i`
`="+ts1.i);`
- `System.out.println("object ts2`
`i="+ts2.i);`
- `}`
- `}`



Understanding Instance and Class Members

```
Test ts1=new Test();
```

```
Test ts2=new Test();
```



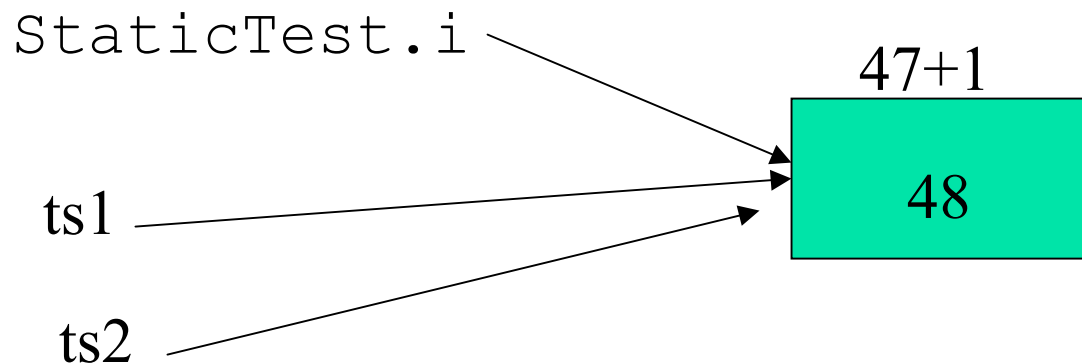


Understanding Instance and Class Members

```
StaticTest.i=StaticTest+1;
```

```
StaticTest ts1=new StaticTest();
```

```
StaticTest ts2=new StaticTest();
```





Understanding Instance and Class Members

- **Methods are similar:** Your classes can have *instance methods* and *class methods*. *Instance methods* operate on the current object's instance variables but also have access to the class variables. *Class methods*, on the other hand, cannot access the instance variables declared within the class (unless they create a new object and access them through the object). Also, class methods can be invoked on the class, you don't need an instance to call a class method.



Understanding Instance and Class Members

- By default, unless otherwise specified, a member declared within a class is an ***instance member***.

The class defined below has one instance variable--an integer named `x`--and two instance methods--`x` and `setX`--that let other objects set and query the value of `x`:



Understanding Instance and Class Members

- `class AnInteger {`
- `int x; public int x()`
- `{ return x; }`
- `public void setX(int newX) {`
- `x = newX;`
- `}`
- `}`



Understanding Instance and Class Members

- ...
- `AnInteger myX = new AnIntegerX();`
- `AnInteger anotherX = new AnIntegerX();`
- `myX.setX(1);`
- `anotherX.x = 2;`
- `System.out.println("myX.x = " + myX.x());`
`System.out.println("anotherX.x = " + anotherX.x());`
- ...



Understanding Instance and Class Members

- What is the output?



Understanding Instance and Class Members

- To specify that a member variable is a class variable, use the **static keyword**.
- `class AnInX`
- `{`
- `static int x;`
- `public int x() {`
- `return x;`
- `}`
- `public void setX(int newX) {`
- `x = newX;`
- `}`
- `}`



Understanding Instance and Class Members

- To specify that a member variable is a class variable, use the **static keyword**.
- `class AnInX`
- `{`
- `static int x;`
- `public int x() {`
- `return x;`
- `}`
- `public void setX(int newX) {`
- `x = newX;`
- `}`
- `}`



Understanding Instance and Class Members

- ...
- `AnInX myX = new AnInX();`
- `AnInX anotherX = new AnInX();`
- `myX.setX(1);`
- `anotherX.x = 2;`
- `System.out.println("myX.x = " + myX.x());`
`System.out.println("anotherX.x = " + anotherX.x()); ...`



Understanding Instance and Class Members

- `class AnIntegerNamedX {`
- `static int x;`
- `public int x() {`
- `return x;`
- `}`
- `public void setX(int newX) {`
- `x = newX;`
- `}`
- `}`

// Does this program correct?



Understanding Instance and Class Members

- //Fix it
- class AnIntegerNamedX {
- static int x;
- static public int x() {
- return x;
- }
- static public void setX(int newX)
- { x = newX;
- }
- }



SingletonPattern

- `//: SingletonPattern.java`
- `// The Singleton design pattern: you can`
- `// never instantiate more than one.`

- `// Since this isn't inherited from a Cloneable`
- `// base class and cloneability isn't added,`
- `// making it final prevents cloneability from`
- `// being added in any derived classes:`



SingletonPattern

- `final class Singleton {`
- `private static Singleton s = new Singleton(47);`
- `private int i;`
- `private Singleton(int x) { i = x; }`
- `public static Singleton getHandle() {`
- `return s;`
- `}`
- `public int getValue() { return i; }`
- `public void setValue(int x) { i = x; }`
- `}`