

Overriding Variables: Shadowing

- We can override methods, can we override instance variables too?
- **Answer:** Yes, it is possible, but **not recommended**
 - Overriding an instance variable is called **shadowing**, because it makes the base instance variables of the base class inaccessible. (We can still access it explicitly using **super.varName**).

```
public class Person {
    extends Person {
        String name;
        // ...
        Staff's name
    }
}

public class Staff
    String name;
    // ... name refers to
```

- This can be **confusing** to readers, since they may not have noticed that you redefined name. Better to just pick a new variable name

Shadowing example

```
class Base {  
    public int x;  
    public Base() {x = 10;}  
    public String foo() {return x+"";}   
}
```

```
class Derived extends Base {  
    public int x;  
    public Derived() { x = 20;}  
    public String foo() {return (x + "\t" + super.x);}  
}
```

```
Derived d = new Derived();  
d.foo();
```

Shadowing example

```
class Base {  
    public int x;  
    public Base() {x = 10;}  
    public String foo() {return x+"";}   
}
```

```
class Derived extends Base {  
    public int x;  
    public Derived() { x = 20;}  
    public String foo() {return (x + "\t" + super.x);}  
}
```

```
Derived d = new Derived();  
d.foo();
```

20 10

Shadowing example

```
class Base {  
    public int x;  
    public Base() {x = 10;}  
    public void foo() {return x;}  
}
```

```
class Derived extends Base {  
    public int x;  
    public Derived() { x = 20;}  
    public void foo() {return (x + "\t" + super.x);}  
}
```

```
Derived d = new Derived();  
Base b = d;  
b.foo();
```

Shadowing example

```
class Base {  
    public int x;  
    public Base() {x = 10;}  
    public void foo() {return x;}  
}
```

```
class Derived extends Base {  
    public int x;  
    public Derived() { x = 20;}  
    public void foo() {return (x + "\t" + super.x);}  
}
```

```
Derived d = new Derived();
```

```
Base b = d;
```

```
b.foo();
```

```
20 10
```

Shadowing example

```
class Base {  
    public int x;  
    public Base() {x = 10;}  
    public void foo() {return x;}  
}
```

```
class Derived extends Base {  
    public int x;  
    public Derived() { x = 20;}  
    public void foo() {return (x + "\t" + super.x);}  
}
```

```
Derived d = new Derived();
```

```
Base b = d;
```

```
d.x;
```

```
b.x;
```

Shadowing example

```
class Base {  
    public int x;  
    public Base() {x = 10;}  
    public void foo() {return x;}  
}
```

```
class Derived extends Base {  
    public int x;  
    public Derived() { x = 20;}  
    public void foo() {return (x + "\t" + super.x);}  
}
```

```
Derived d = new Derived();
```

```
Base b = d;
```

```
d.x; 20
```

```
b.x; 10
```

super and this

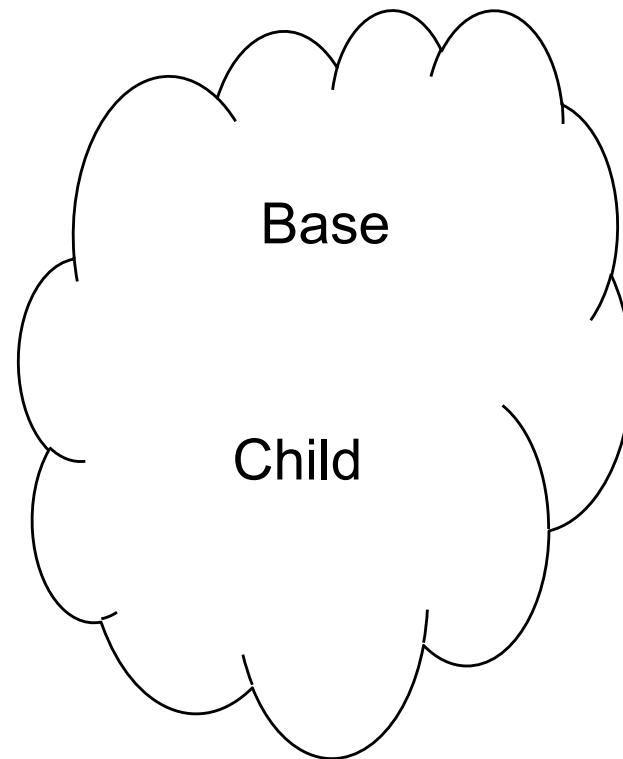
- **super**: refers to the base class object
 - We can invoke any base class constructor using **super(...)**.
 - We can access data and methods in the base class (Person) through **super**. E.g., toString() and equals() invoke the corresponding methods from the Person base class, using **super.toString()** and **super.equals()**.
- **this**: refers to the current object
 - We can refer to our own data and methods using “**this**.” but this usually is not needed
 - We can invoke any of our own constructors using **this(...)**. As with the super constructor, this can only be done **within a constructor**, and must be the **first statement** of the constructor. Example:

```
public Fraction(int n) {  
    this(n,1);  
}
```


Memory Layout

```
class Base{  
    private int a;  
    protected int b;  
    protected int c;  
    protected void m1(){}  
    public void m2(){}  
}
```

```
class Child extends Base{  
    private int d;  
    public void m1(){}  
    public void m3(){}  
}
```



The Java Virtual Machine does not mandate any particular internal structure for objects.

Memory Layout

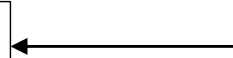
```
class Base{  
    private int a;  
    protected int b;  
    protected int c;  
    protected void m1() {}  
    public void m2() {}  
}
```

VTABLE

Pointer to m1()
Pointer to m2()

Base object

Pointer to vtable
a
b
c



```
class Child extends Base{  
    private int d;  
    public void m1() {}  
    public void m3() {}  
}
```

Memory Layout

```
class Base{
  private int a;
  protected int b;
  protected int c;
  protected void m1() {}
  public void m2() {}
}
```

```
class Child extends Base{
  private int d;
  public void m1() {}
  public void m3() {}
}
```

VTABLE

Pointer to m1()
Pointer to m2()

Base object

Pointer to vtable
a
b
c

VTABLE

Pointer to m1()
Pointer to m2()
Pointer to m3()

Child object

Pointer to vtable
a
b
c
d

Memory Layout

```
class Base{  
    private int a;  
    protected int b;  
    protected int c;  
    protected void m1() {}  
    public void m2() {}  
}
```

VTABLE

Pointer to m1()
Pointer to m2()

Base object

Pointer to vtable
a
b
c

```
class Child extends Base{  
    private int d;  
    public void m1() {}  
    public void m3() {}  
}
```

VTABLE

Pointer to m1()
Pointer to m2()
Pointer to m3()

Child object

Pointer to vtable
a
b
c
d

Each class has one vtable.

All objects of the this class shares the vtable.

Inheritance and Private

- **Private members:**
 - Child class **inherits all the private data** of Base class
 - However, **private members** of the base class **cannot** be accessed directly
- **Why is this?** After you have gone to all the work of setting up privacy, it wouldn't be fair to allow someone to simply **extend** your class and now have access to all the **private** information

Quiz 5: True/False

Excepting Object, which has no superclass, every class has one and only one direct superclass.

- A. True
- B. False

Quiz5: True/False

Excepting Object, which has no superclass, every class has one and only one direct superclass.

- A. True
- B. False

Quiz 6:

```
class Base {
    public void foo() {
        println("Base");
    }
}
class Derived extends Base {
    private void foo() {
        println("Derived");
    }
}
...
Base b = new Derived();
b.foo();
```

- A. Base
- B. Derived
- C. Compiler Error
- D. Runtime Error

...

Quiz 6:

```
class Base {
    public void foo() {
        println("Base");
    }
}
class Derived extends Base {
    private void foo() {
        println("Derived");
    }
}
...
Base b = new Derived();
b.foo();
```

- A. Base
- B. Derived
- C. Compiler Error
- D. Runtime Error

It is compiler error to give more restrictive access to a derived class function which overrides a base class function.

Quiz 7:

class Animal has a subclass Mammal. Which of the following is true:

- A. Because of single inheritance, Mammal can have no subclasses.
- B. Because of single inheritance, Mammal can have no other parent than Animal.
- C. Because of single inheritance, Animal can have only one subclass.
- D. Because of single inheritance, Mammal can have no siblings.

Quiz 7:

class Animal has a subclass Mammal. Which of the following is true:

- A. Because of single inheritance, Mammal can have no subclasses.
- B. Because of single inheritance, Mammal can have no other parent than Animal.
- C. Because of single inheritance, Animal can have only one subclass.
- D. Because of single inheritance, Mammal can have no siblings.

Access level

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N