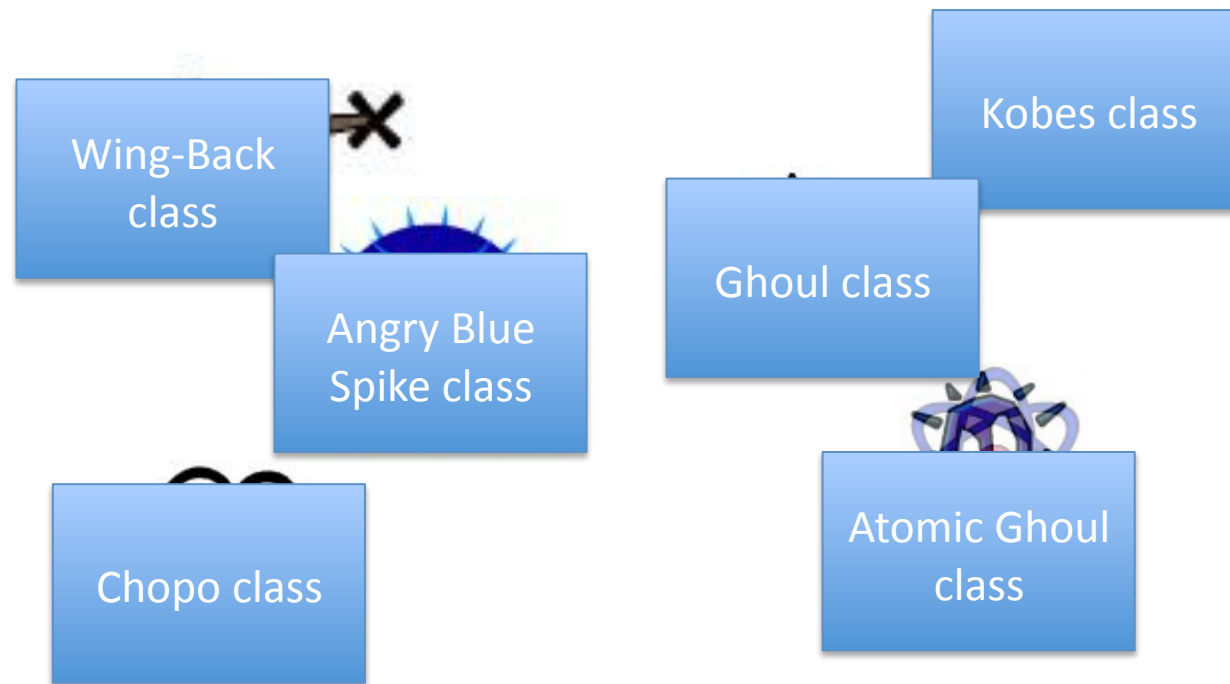
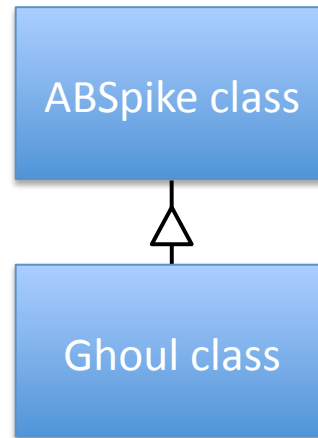


Game Characters



Characters in a game



Inherited methods

```
out.println(ScaryHarry.getLife());
```

```
if (ScaryHarry.isCaptured()) { ...
```

*These methods are **inherited** from
the super class, **ABSpoke***

Constructors

ABSpike class

- color: input parameter
- captured: will be set to default of false
- life: set default value to 10
- position: input parameter

Ghoul class

- color: input parameter
- captured: will be set to default of false
- life: set default value to 15
- position: input parameter

Constructors

ABSpike class

- color: input parameter
- captured: will be set to default of false
- life: set default value to 10
- position: input parameter

Ghoul class

- call constructor from Character
- life: set default value to life + 5

Ghoul class Constructors

- call super(team, starting position);
- life = 5;  life + 5

```
private void setLife(int change){  
    life = life + change;  
}
```

setLife(int) in ABSpike super class

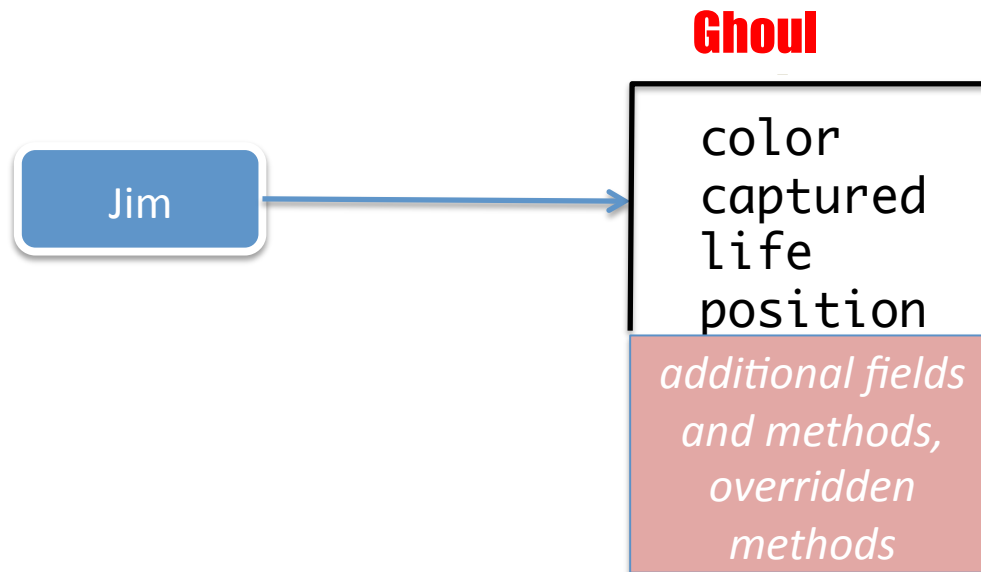
```
protected void setLife(int change){  
    life = life + change;  
}
```

- ✓ **Visible to the class it is defined in**
- ✓ **Visible to all subclasses**

Ghoul class

```
public class Ghoul extends ABSpike{  
    public Ghoul(char team, int[] startingPos){  
        super(team, startingPos);  
        setLife(5); //life set to default + 5  
    }  
  
    public boolean move (int x, int y){  
        if(1 == Math.abs(getX() - x) && 1 == Math.abs(getY() - y)){  
            return(super.move(x, y));  
        }  
        return(false);  
    }  
}
```


Constructors in a subclass



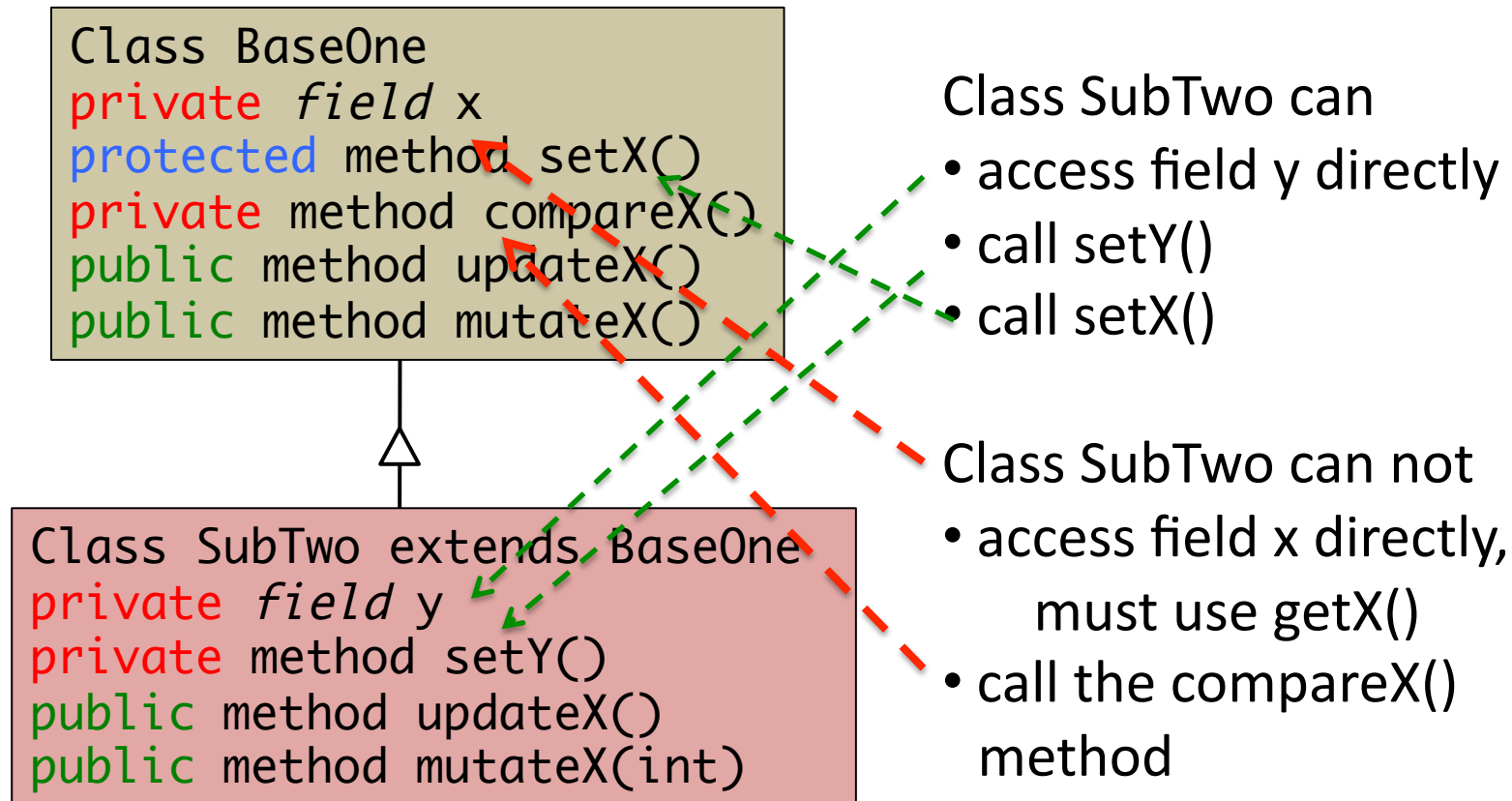
Constructors in a subclass

- Call super constructor first
- If no call to a super class constructor is made
 - an *implicit* call to default constructor in super class is made
 - if there is no default constructor in the super class, another constructor must *explicitly* be called

Private methods and fields

- Are not accessible outside the class where they are defined or declared
- Are not accessible to subclasses, *even though* they are inherited
- Subclasses must use public or protected methods to access their own fields

The relationship between a super class and its subclass(es)



The relationship between a super class and its subclass(es)

```
Class BaseOne
private field x
protected method setX()
private method compareX()
public method updateX()
public method mutateX()
```



```
Class SubTwo extends BaseOne
private field y
private method setY()
public method updateX()
public method mutateX(int)
```

a client program...


```
BaseOne user3 = new BaseOne();
SubTwo user17 = new SubTwo();

for (int i:= 0; i<k; i++){
    user3.updateX();
}


user17.mutateX(8);
user17.y = 874;
user3.compareX(user17);
```

The relationship between a super class and its subclass(es)

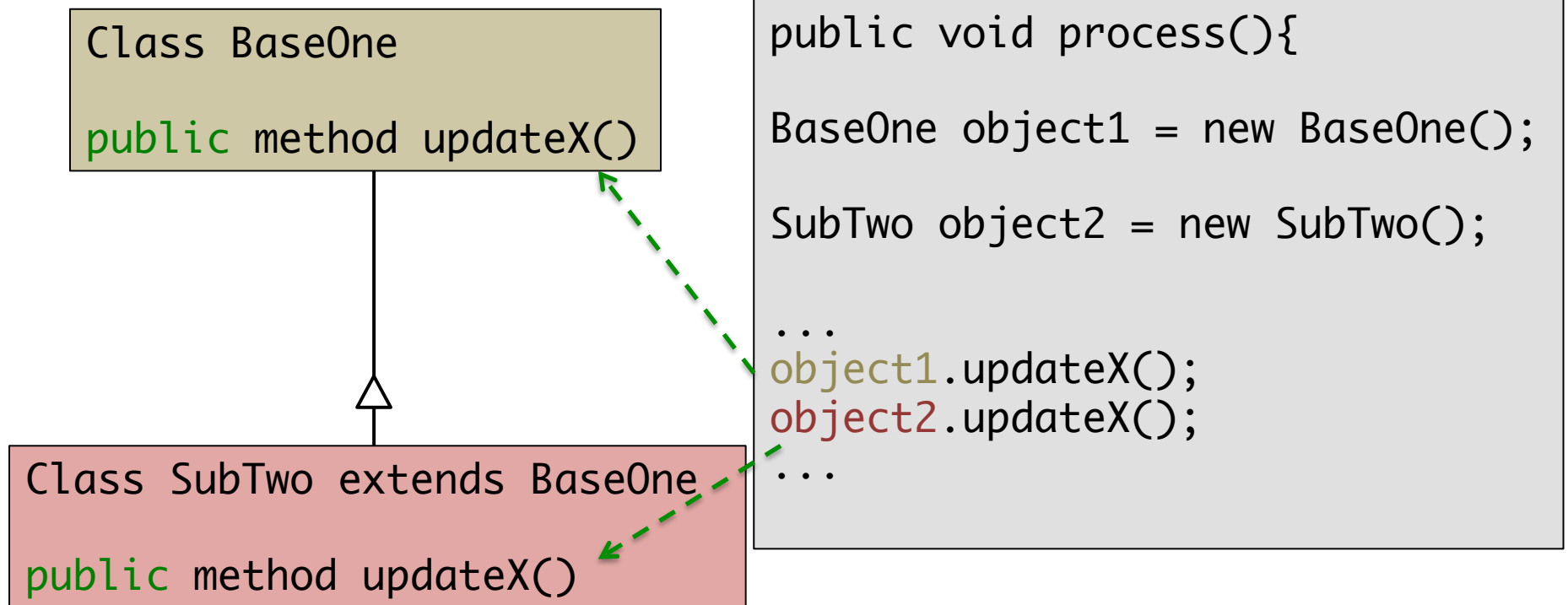
```
Class BaseOne  
private field x  
protected method setX()  
private method compareX()  
public method updateX()  
public method mutateX()
```



```
Class SubTwo extends BaseOne  
private field y  
private method setY()  
public method updateX()  
public method mutateX(int)
```




The relationship between a super class and its subclasses




The relationship between a super class and its subclass(es)

```
Class BaseOne  
private field x  
protected method setX()  
private method compareX()  
public method updateX()  
public method mutateX()
```



```
Class SubTwo extends BaseOne  
private field y  
private method setY()  
public method updateX()  
public method mutateX(int)
```



The relationship between a superclass and its subclasses

