# Inheritance

# Inheritance

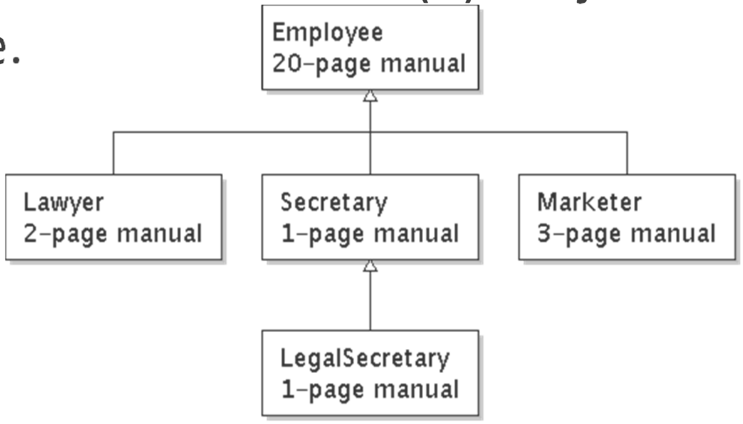
## Definitions

= way of forming new classes based on existing ones

= way to share/**reuse code** between two or more classes

## Terminology

- **superclass**: Parent class being inherited from / extended / specialized.
- **subclass**: Child class that inherits behavior from superclass.
  - gets a copy of every field and method from superclass
- **is-a relationship**: Each object of the subclass also "is a(n)" object of the superclass and can be treated as one.



## Inheritance syntax

```
public class NameofSubClass extends NameOfSuperclass
{
```

Example

```
public class Lawyer extends Employee {
    ...
}
```

By extending `Employee`, each `Lawyer` object now:

- receives a copy of each method / field from `Employee` automatically
- can be treated as an `Employee` by client code
- Lawyer can also replace ("override") behavior from Employee.

## Let’s look more into Overriding

**Definition**

- To write a new version of a method in a subclass that replaces the superclass's version
- No special syntax required to override a superclass method. Just write a new version of it in the subclass.

```
public class Lawyer extends Employee {
    // overrides getVacationForm in Employee class

    public String getVacationForm() {
        return "pink";
    }
    ...
}
```

## Let’s look more into Overriding

**Definition**

- To write a new version of a method in a subclass that replaces the superclass's version
- No special syntax required to override a superclass method. Just write a new version of it in the subclass.

```
public class Lawyer extends Employee {
    // overrides getVacationForm in Employee class
    @override
    public String getVacationForm() {
        return "pink";
    }
    ...
}
```

stack overflow [https://stackoverflow.com/questions/94361/when-do-you-use-javas-override-annotation-and-why](https://stackoverflow.com/questions/94361/when-do-you-use-javas-override-annotation-and-why)

## How do subclasses use superclass' methods?

Subclasses' methods may use superclasses' methods/constructors:

```
super.method(parameters)          // method
super(parameters);                // constructor


public class Lawyer extends Employee {
    public Lawyer(String name) {
        super(name);
    }

    // give Lawyers a $5K raise (better)
    public double getSalary() {
        double baseSalary = super.getSalary();
        return baseSalary + 5000.00;
    }
}
```

## How do Subclasses use superclass' fields?

THEY DON'T

Rules =

- Subclasses are not allowed to use superclass' private fields
  - i.e. Inherited private fields/methods cannot be directly accessed by subclasses
  - *aka The subclass has the field, but it can't touch it*

```
public class Employee {
    private double salary;
    ...
}

public class Lawyer extends Employee {
    ...
    public void giveRaise(double amount) {
        salary += amount;   // error; salary is private
    }
}
```

? How can we allow subclasses to access / modify these fields?

## Solution = Protected fields/methods

**protected fields** or **methods** may be seen/called only by:

- the class itself, its subclasses, other classes in same "package"

Syntax

```
protected type name;     // field
protected type name(type name, ..., type name) {
    statement(s);        // method
}
```

Example

```
public class Employee {
    protected double salary;
    ...
}
```

## Inheritance and constructors

Problem

- IF we replace our constructor w/o parameters w/ a constructor that requires parameters in `Employee`
- THEN our subclasses do not compile;

```
Lawyer.java:2: cannot find symbol
symbol  : constructor Employee()
location: class Employee
public class Lawyer extends Employee {
       ^
```

Solution

- IF we write a constructor (that requires parameters) in the superclass
- THEN must now rewrite constructors for our employee subclasses

## Let's dig a bit deeper on this…

Rules = Constructors are not inherited

- Subclasses don't inherit the `Employee(int)` constructor.
- Subclasses receive instead a default constructor that contains:

```
public Lawyer() {
    super();        // calls Employee() constructor
}
```

But our `Employee(int)` replaced the default `Employee()`.

- The subclasses' default constructors are now trying to call a non-existent default `Employee` constructor.

## How do we refer to the superclass constructors?

Syntax

```
super(parameters);
```

Example

```
public class Lawyer extends Employee {
    public Lawyer(int years) {
        super(years);  // calls Employee c'tor
    }
    ...
}
```

Rules – The `super` call must be the first statement in the constructor