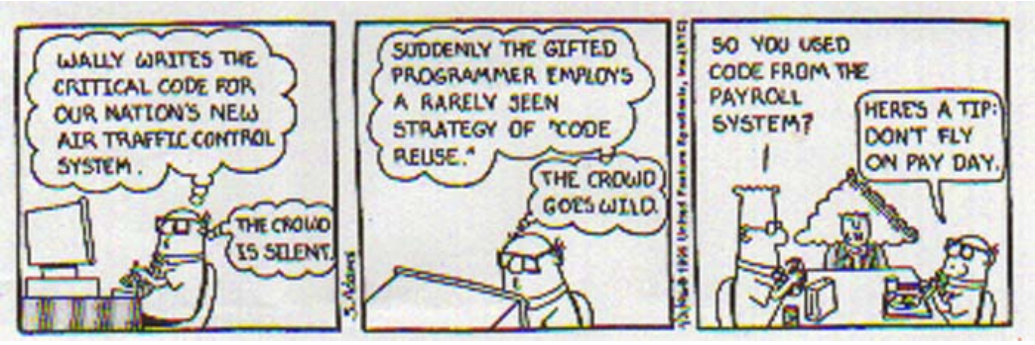
Name: _____

AP Computer Science A
Inheritance Basics

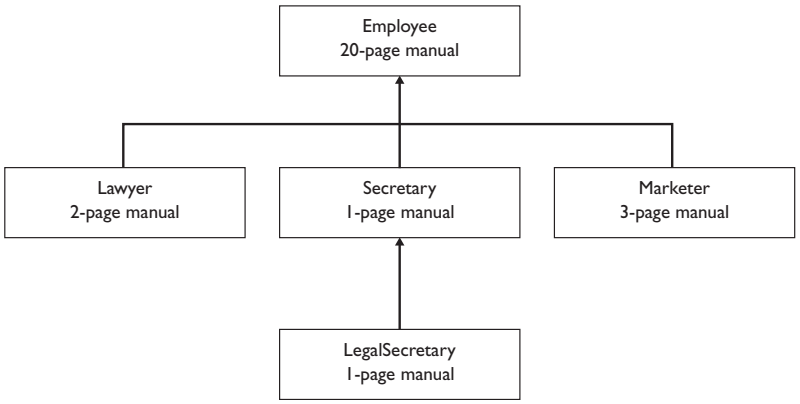
1970's Code Was Bad

	Projects became more complex and frequently would go over budget
	More complex projects also had more bugs and were more likely to malfunction
	Adding more people to projects made the problems WORSE (See: The Mythical Man Month by Fred Brooks)
	Roughly 90% of time spent on projects was maintaining legacy code, only 10% on new code. Old code was commonly disorganized, redundant (ex: early GUI code), poorly commented, etc. Updating code generally lead to new bugs.

Code Reuse -



Hierarchies



Law Firm Employees
lawyers, general secretaries, legal secretaries, and marketers

Work Rules
vacation and sick days, medical benefits, harassment regulations,

The Manuals
Each employee gets the 20-page manual, and each type of employee gets an addendum (some new rules, some rules changed)

Why would it be a worse idea to create a 22 page manual for Lawyers, a 21 page manual for Secretaries, etc?

General Hierarchy Ideas

1) It's useful to be able to specify a broad set of rules that will apply to many related groups (the 20-page manual).
2) It's also useful to be able to specify a smaller set of rules specific to a particular group, and to be able to replace some rules from the broad set
3) Employee's in the Hierarchy follow an ____-__ relationship

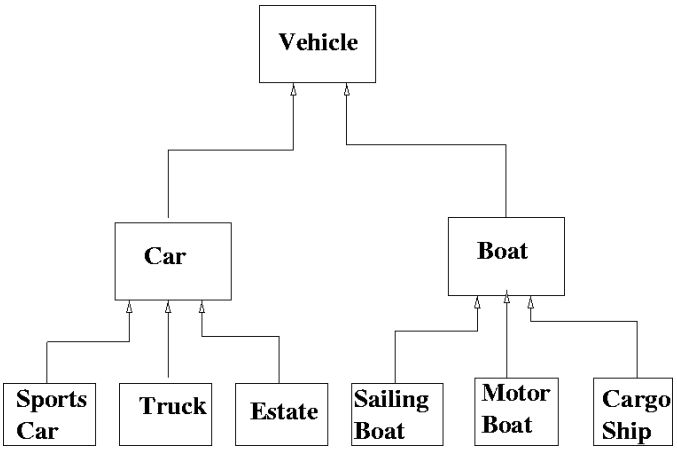
Is-A Relationship -

Example: Lawyer Is-A _____. LegalSecretary Is-A _____ and a _____.

If a class A has an IS-A relationship with class B, then class A can fill in for class B.

Example: If a Secretary gets sick, then a _____ can fill in for them.

Inheritance Hierarchy -



Extending a Class

The Situation

Employees	Hours / Week	Salary	Weeks Paid Vacation	Vacation Form	Special
Lawyer	40	$40,000	3	Pink	Handle Lawsuit
Secretary	40	$40,000	2	Yellow	Take Dictation
Legal Secretary	40	$45,000	2	Yellow	File Legal Briefs
Marketer	40	$50,000	2	Yellow	Advertise

Bad Java Code

```
public class Employee {
    public int getHours() {
        return 40;
    }

    public double getSalary() {
        return 40000.0;
    }

    public int getVacationDays() {
        return 10;
    }

    public String getVacationForm() {
        return "yellow";
    }
}
```

```
public class Secretary {
    public int getHours() {
        return 40;
    }

    public double getSalary() {
        return 40000.0;
    }

    public int getVacationDays() {
        return 10;
    }

    public String getVacationForm() {
        return "yellow";
    }

    // this is the only added behavior
    public void takeDictation(String text) {
        System.out.println("Dictating text: " + text);
    }
}
```

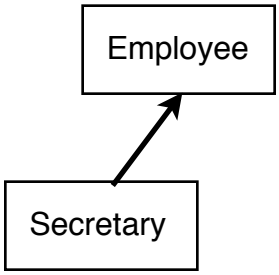
What We'd Like To Do

```
public class Secretary {
    copy all the methods from the Employee class.

    // this is the only added behavior
    public void takeDictation() {
        System.out.println("I know how to take dictation.");
    }
}
```

We want the Secretary class to "get" all the methods from the Employee class - we accomplish this with _____.

Some Vocab



Inheritance - technique that allows a derived class to extend the functionality of a base class, inheriting all of its state and behavior.

Superclass (parent / base) -

Subclass (child / derived) -

Extends -

Single Inheritance -

General Java Code

```
public class <name> extends <superclass> {
    ...
}
```

Secretary Class Using Inheritance

```
public class Secretary extends Employee {
    public void takeDictation(String text) {
        System.out.println("Dictating text: " + text);
    }
}
```

Client Code

```
public static void main(String[] args) {
        Syso(“Employee: ”);
        Employee theo = new Employee();
        Syso(theo.getHours() + “, ” + theo.getSalary() + “, “ + theo.getVacationDays() + “, ” + theo.getVacationForm());
        System.out.println();

        Secretary tracy = new Secretary();
        Syso(tracy.getHours() + “, ” + tracy.getSalary() + “, “ + tracy.getVacationDays() + “, ” + tracy.getVacationForm());
        tracy.takeDictation(“Hello World!”);
        System.out.println();
}
```

Output:

Overriding Methods

The Secretary class only needed to add new functionality, but the other classes (Marketer, Lawyer, LegalScretary) need to change inherited functionality. This is accomplished by override the inherited methods.

Override -

```
public class Lawyer extends Employee {
    // overrides getVacationDays from Employee class
    public int getVacationDays() {
        return 15;
    }

    // overrides getVacationForm from Employee class
    public String getVacationForm() {
        return "pink";
    }

    // this is the Lawyer's added behavior
    public void sue() {
        System.out.println("I'll see you in court!");
    }
}
```

```
public class LegalSecretary extends Secretary {
    // overrides getSalary from Employee class
    public double getSalary() {
        return 45000.0;
    }

    // new behavior of LegalSecretary objects
    public void fileLegalBriefs() {
        System.out.println("I could file all day!");
    }
}
```

Client Code

```
public static void main(String[] args) {
        Syso("Lawyer: ");
        Lawyer ali = new Lawyer();
        Syso(ali.getHours() + ", " + ali.getSalary() + ", " + ali.getVacationDays() + ", " + ali.getVacationForm());
        ali.sue();
        System.out.println();

        LegalSecretary tracy = new LegalSecretary();
        Syso(bryce.getHours() + ", " + bryce.getSalary() + ", " + bryce.getVacationDays() + "," + bryce.getVacationForm());
        bryce.takeDictation("cool");
        bryce.fileLegalBriefs();
        System.out.println();
}
```

Output:

Super Type Variables / Sub Type Data - A Okay! (reverse, not so much)

Employee e1 = new Employee(); Employee e2 = new Lawyer(); Employee e3 = new Secretary();

Secretary s1 = new Employee(); Secretary s2 = new Secretary(); Secretary s3 = new LegalSecretary();

LegalSecretary ls1 = new LegalSecretary(); LegalSecretary ls2 = new Secretary();