# Create Employee v2.00

1. Create a NEW project named Employee
2. Add a java class named **Main** to the src folder and use psvm to create a main method
3. Review the specification
   1. Employees work 40 hr/wk
   2. Employees make $40,000 per year
      1. Except for legal secretaries who make $5,000 extra per year
      2. Except for marketers who make $10,000 extra per year
   3. Employees have 2 weeks of paid vacation per year
      1. Except for lawyers who get an extra week
   4. Employees should use a yellow form to apply for leave
      1. Except for lawyers who use a pink form
   5. Each type of employee has some unique BEHAVIOR
      1. Lawyers know how to sue
      2. Marketers know how to advertise
      3. Secretaries know how to take dictation
      4. Legal Secretaries know how to prepare legal documents
4. Add a java class named **Employee** that implements the STATE and BEHAVIORS of a general employee
   1. getHours() - returns 40 (hours per week)
   2. getSalary() - returns 40,000 (dollars per year)
   3. getVacationDays() - returns 10 (days per year of paid vacation – 2 weeks)
   4. getVacationForm() - returns “yellow” (use yellow form)
5. Cannot run Employee class directly…must use the Main client
   1. Can explain as a person using a tool. A hammer cannot hammer by itself. A class cannot run itself…it must be run by a client.
   2. Create a new Employee instance and demo the methods
      1. System.out.println(“Employee works “ + e.getHours() + “ per week”);
      2. etc.
6. Add a java class named **Secretary** (copy from Employee to save time)
   1. Initial version implements Secretary STATE and BEHAVIOR without inheritance
      1. getHours() - returns 40 (hours per week)
      2. getSalary() - returns 40,000 (dollars per year)
      3. getVacationDays() - returns 10 (days per year of paid vacation – 2 weeks)
      4. getVacationForm() - returns “yellow” (use yellow form)
      5. **takeDictation(String text) - returns “Taking dictation of text: “ + text**
   2. This is REDUNDANT because the Employee class already implements all but the takeDictation method.
   3. Use INHERITANCE to allow the Secretary class to INHERIT the BEHAVIOR from the Employee class.
      1. Child class gets a COPY of every field and method from superclass
      2. public class Secretary extends Employee {…}
      3. This means that Secretary automatically gets all the Employee methods (don’t have to explicitly implement them).
      4. Secretary instances can be treated as an Employee by client code (will demo later)
      5. Only need to keep the fields and methods (STATE and BEHAVIOR) that are UNIQUE to Secretary class (i.e., takeDictation).
7. Add a java class named **Lawyer** (copy from Employee)
   1. Have Lawyer class extend Employee class and remove Employee methods
      1. public class Lawyer extends Employee { … }
   2. According to spec
      1. Lawyers get an extra week of vacation (total of 3 weeks or 15 days)
      2. Lawyers use the pink vacation form
      3. Lawyers know how to sue
   3. PROBLEM: we want lawyers to inherit MOST (but not all) Employee behaviors – we want to REPLACE some of the Employee behaviors with NEW behaviors
   4. SOLUTION: use the OVERRIDE feature to override a parent method in a child class
      1. Override getVacationForm
      2. Just having the method in the child class overrides the parent class method (even if it does the same thing)…so be sure to delete it if you truly want to inherit the parent class behavior
      3. Make sure that COMMENTS identify the intent to override (so future coders don’t remove it)
   5. Add method public void sue() { System.out.println(“I’ll see you in court!”);
   6. NOTE: it is a good idea to use the @Override “annotation” (sometimes called a “decorator”) to identify methods being overridden.
      1. If you create the overridden methods using the CNTL-O hotkey (with cursor on the class being extended) then the @Override decorator is added automatically.
      2. Two reasons to use the annotation
         1. If the annotated method is not actually overriding anything then the compiler will isuse a warning
         2. It helps make the source code more READABLE
8. Add a java class named **Marketer** (copy from Employee or maybe from Lawyer - smile)
   1. Have Marketer class extend Employee
   2. Add advertise method – System.out.println(“Act now while supplies last!”);
   3. Override getSalary to return 50000
9. Add a java class named **LegalSecretary** that extends Secretary (instead of Employee)
   1. Override getSalary() to return 45000
   2. Add fileLegalBriefs BEHAVIOR (public void fileLegalBriefs() {…})
      1. System.out.println(“I could file all day!”);
   3. Inherits from SECRETARY so it gets all the fields and behaviors of Secretary
10. DEMO using Main.java from the KEY Employee v2.00

# Create Employee v4.00

1. Change happens
   1. Everyone gets a 10000 per year raise
      1. Base is now 50000
      2. Legal Secretaries get 55000
      3. Marketers get 60000
2. We must modify the code to make this change. How many classes do we have to edit?
   1. Modify the Employee class.
   2. ARE WE FINISHED??? (ans: NO – because subclasses are still incorrect because they have overridden getSalary – prove by demo)
   3. We could fix things by updating the getSalary method in the two subclasses (LegalSecretary and Marketer)…BUT THIS SEEMS WRONG…future changes require us to touch multiple files…IS THERE A BETTER WAY??? Yes…
   4. The subclass salaries do depend on the Employee salary but we are not currently reflecting that…how can we do that?
   5. Use the **super** keyword
   6. LegalSecretary salary is super.getSalary() + 5000.0 (do in two steps…baseSalary = super.getSalary(); and return baseSalary + 5000.0)
   7. Marketer salary is super.getSalary() + 10000.0
   8. Update ALL other child classes to simply return the baseSalary
3. DEMO using Main.java from the KEY Employee v4.00

# Create Employee 5.00

1. Change happens (again)
   1. Want to give vacation days based on years worked (for each year worked they get 2 additional days)
      1. EXCEPT Secretaries whose years of employment are NOT TRACKED and they do NOT earn any extra vacation days for years worked
   2. When an Employee object is CREATED (“constructed”) it should require the number of years the person has been with the company
   3. Add STATE to hold years in the Employee class (private int years;)
   4. Add a constructor to populate the years field
   5. Update getVacationDays method to return 10 + 2 \* years days
2. This WILL NOT COMPILE…because once we write a constructor that requires parameters in the super class there is NO DEFAULT CONSTRUCTOR in the super class…AND…default constructors for child classes call the default constructor for the super class (which no longer exists)…SEE SLIDE
3. Let’s add the default constructors back to Employee …
   1. For Employee add the default constructor and call the constructor using years with zero years as the parameter.
   2. This fixes the compile errors BUT when we instantiate a Secretary object the value for years is NEVER SET (well it is an int so it initializes as zero which is OK in this case, but in general not setting the state will probably be a bad thing). How do we fix this?
4. We can add contructors to our child classes that take a parameter for years but HOW do we pass that to the parent class state? We could just set years in the child constructors…BUT…this is the NOT the best. The BEST way to set the STATE is to call the super class constructor so we don’t miss any logic that might be contained there.
   1. We can call a super class constructor just using super(parameters) to call the constructor with the matching signature.
5. Do this for the Lawyer and Marketer class
6. But secretaries don’t get this benefit. The secretary class only needs a default constructor since we never track years. That default constructor can calls the super constructor with zero years.
7. Since secretary has a default constructor the LegalSecretary class doesn’t need a constructor.
8. DEMO using Main.java from the KEY Employee v5.00

# Create Employee 6.00

1. Change happens (again)
   1. Lawyers should now get $5000 more salary for each year at the company
   2. If we use super.getSalary() + 5000 \* years then we get an ERROR because we used the private modifier in Employee for years (Lawyer cannot SEE the field named years)
   3. This is INTENTIONAL so that someone can’t break ENCAPSULATION by creating a sub class (that could be a security problem).
   4. To get around this add a “getter” for years to Employee
      1. public int getYears() { return years; }
      2. In general, add an accessor method (a “getter”) for any field needed by a subclass
   5. Then use getYears in Lawyer…
2. DEMO using Main.java from the KEY Employee v6.00

# Create Employee 7.00

1. Secretary currently has a BAD solution because we are NOT tracking years. What happens if we want to give secretaries some other benefit based on years? If we set years then their salary gets adjusted.
2. Let’s redesign Employee to allow for a better solution…SEPARATE the standard 10 vacation days from those awarded based on seniority
   1. Update Employee getVacationDays() to add getSeniorityBonus()
   2. Add a getSeniorityBonus() method to return 2 \* years;
   3. How does this help with Secretary?
   4. We can now OVERRIDE getSeniorityBonus to return 0 for Secretaries
   5. Then we can have a Secretary constructor that defines years
3. DEMO using Main.java from the KEY Employee v7.00

# GO TO SLIDES on POLYMORPHISM