Object oriented programming

- **Pre-OOP**
  - Programmer's duty to devise and enforce logical groupings of the data types and the functions that operate on them.

- **OOP**
  - Functionality is officially and tightly grouped with the type that it operates on.
  - Code re-use
  - OOP uses "messages" instead of function calls.
  - What is a class?

  - What is an object?
Inheritance

- **Process by which a class obtains the properties of its superclasses (ancestors).**
- **Hierarchy**
  - Classes are generally structured as a tree
  - Subclasses below - Superclasses above
- **Overriding**
  - When an object receives a message, it checks its own methods first before consulting its superclass.
  - First matching method found in the hierarchy overrides all methods of the superclass(es)
- **Polymorphism**
  - Many classes will respond to a single message
  - In a graphics program, many of the classes are likely to implement the method `drawSelf()`.
Problem Outline

- Model Employee Performance at Springfield’s Nuclear Power Plant.
Worker Types

- 4 worker types: Grunt, Homer, Manager, and Boss

- All workers have a blood pressure, which tends to go up with excessive work.

- Grunts and Homers keep track of how many years they have until retirement. The closer until retirement, the happier they tend to be.

- Managers have underlings they order around--their productivity is based upon that of their underlings.
Worker Methods

- All workers respond to three messages:
  - Stress
  - Productivity
  - Work
double stress();

- Returns the current stress level of the employee.
- Stress can be positive (stressing) or negative (relaxing).
  - For all employees, the underlying stress level is (BloodPressure - 50) but it can never go over 200.
  - The stress level of Managers and Bosses is the square of the underlying stress.
  - The stress level of Grunts and Homers is the underlying stress plus the number of years until retirement.
double productivity();

Returns the current productivity factor of an employee which indicates how much work they can accomplish per hour. The productivities of Grunts and Homers are:

Grunt: 1.0 + years-to-retire
Homer: 1.0/(years-to-retire + 0.1)

Managers tend to get a lot of useful work out of their underlings, so the productivity of a Manager is the sum of the productivities of that manager's underlings. Bosses tend to mostly scare their underlings to death, so the productivity of a Boss is only the mean average of the productivities of that Boss's underlings.
bool work(double amount);

- Sent to an employee when that employee has been asked to do work. The method returns true if the work was done, and false otherwise.

- Employees refuse work and return false if their Stress level is greater than 100 or the amount of work is > 10 times their productivity.

- Passing that, Grunts and Homers just do the work (and their blood pressures are raised by their Productivity).
bool work(double amount); (continued)

- Managers and Bosses respond by delegating the work to their underlings. Managers ask each underling in turn to do the work until one of the underlings agrees.

- At that point, the manager is satisfied and returns true, confident that the work has been done. The manager returns false if no underling will do the work.

- Bosses work similarly, but when an underling does the work, there's only a 50% chance that the Boss will be satisfied. If the boss is satisfied, they return true. If the boss is not satisfied, they go ahead and ask more underlings to do the work, even though at least one underling has already done it. The Bosses return false if they were never satisfied.
Class hierarchy

- Employee
  - Worker
    - Manager
    - Homer
  - Manager
    - Boss

[ ] indicates abstract Class
Class Outlines

- **Employee**: Abstract base class. All methods implemented but `productivity()`, which is abstract.

- **Worker**: Abstract class, extends Employee. Refines `stress()` and `work()`.

- **Homer and Grunt**: Extends Worker. Implements `productivity()`

- **Manager**: Extends Employee. Refines `stress()` and implements `productivity()`

- **Boss**: Extends Manager. Refines `productivity()`
Problem for exam

- Define the Employee class, subclasses, and header files
  - Do you need any more information to successfully create these classes?

- How would you create a group of random employees?