Object-Oriented Programming
Imperative vs. Object-Oriented Programming

Imperative Programming (50s - 70s)
- uses statements to change a program's state
- a variable contains a value, an assignment statement changes it
- a print statement sends a value to output
- a "go to" transfers control to another statement
- "verbs" are the most important thing
- Functions / subprograms not really emphasized - used for reuse
- Examples: FORTRAN, COBOL, Algol, Basic

Object-Oriented Programming (80s - present)
- Imperative in style, structured, but features added to support "objects"
- Ex: Smalltalk, Simula, C++, Python, Visual Basic, Java, Ruby
Calculating pay in FORTRAN

```fortran
READ(5,6) HRS, RATE
6 FORMAT(I3,F6.2)
IF (HRS.LE.40) GO TO 5
OVTPAY = (HRS - 40) * RATE * 1.5
WRITE(6,7) OVTPAY
7 FORMAT("OVERTIME PAY IS",F6.2)
REGPAY = OVTPAY + 40 * RATE
GO TO 8
5 REGPAY = HRS * RATE
8 WRITE(6,9) REGPAY
9 FORMAT("REGULAR PAY IS",F6.2)
STOP
END
```
In Imperative Programming, you:
• break down a task into variables, data structures, and subprograms
• you use subprograms to operate on data structures

In Object-Oriented Programming, you:
• define objects that expose behavior (methods) and data (attributes) using well-defined interfaces
• bundle everything together, so that an object only operates on its own attributes using methods
Four Basic Programming Concepts in OOP:

- **Encapsulation**: hiding implementation details of a class from other objects.

- **Abstraction**: simplifying complex reality by modeling classes appropriate to the problem.

- **Inheritance**: a way to define new classes using parts of classes that have already been defined.

- **Polymorphism**: the process of interpreting an operator or function in different ways for different data types.
<table>
<thead>
<tr>
<th>class variables</th>
<th>instance variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared by all instances of the class.</td>
<td>&quot;Owned&quot; by each individual instance in a class. Each instance has its own copy.</td>
</tr>
<tr>
<td>All instances <em>of that class</em> can access it.</td>
<td>Only the instance can access its instance variables.</td>
</tr>
<tr>
<td>When changed, the change is reflected by whichever</td>
<td>When changed, only the instance is affected.</td>
</tr>
<tr>
<td>instance tries to access it.</td>
<td></td>
</tr>
</tbody>
</table>