

*Core Java Course*  
*Objects and Classes in Java*  
(Chapter 4)  
Prof. Jeffrey Blessing  
MSOE

---

---

---


---

---

---

---

---



*Agenda*

- ❖ Introduction to Object Oriented Programming
- ❖ Implementation of a class in Java
- ❖ How to write reusable classes

2

---

---

---


---

---

---

---

---



*Little Quiz*

- ❑ What is a class?
- ❑ Why do we like objects so much?
- ❑ Inheritance and polymorphism around us (examples)

3

---

---

---

---

---

---

---

---

*Structured Programming*

- ❑ Find an algorithm to solve the problem
- ❑ Define appropriate data structures
- ❑ Break program into small, logically related procedures.
- ❑ Program needs to run only when the actual calculation is performed
- ⇨ Example: Calculating  $Y=X^2$

4

---

---

---

---

---

---

---

*Object-Oriented Programming*

- ❖ Pre-packaged functionality
  - Influenced by hardware component approach
  - ⇨ Example: Computer store, Legos
- ❖ Every object knows how to handle its internal state and how to interface with other objects
- ❖ Program runs for the life of objects
  - ⇨ Example: A Window object on PC screen

5

---

---

---

---

---

---

---

*Vocabulary (cont.)*

- ❑ **Class** - a template from which similar objects are made (cookie cutter)
- ❑ **Instance** - an object created from a class
- ❑ **Attribute** - a parameter to characterize the state of an object
- ❑ **Method** - a way to perform a task on object
- ❑ **Extend** class - create a class that inherits properties and methods of its parent

6

---

---

---

---

---

---

---

*Objects*

- ❖ Distinct behavior
  - Same behavior for all objects of a class
- ❖ Defined state
  - State depends on values of instance variables
- ❖ Unique identity
  - Reference to an object like ID#, license#, or an "object" variable

7

---

---

---

---

---

---

---

*Encapsulation*

- ❖ Combining data and behavior in one package
- ❖ Behavior is defined by access methods provided
- ❖ Internal object data is hidden
- ❖ Object state can be changed only by using a method

8

---

---

---

---

---

---

---

*Relationships between objects*

- ❖ Inheritance
  - B extends A;
- ❖ Association
  - A is aware of B
- ❖ Use
  - A uses services provided by B
- ❖ Containment
  - A contains B

9

---

---

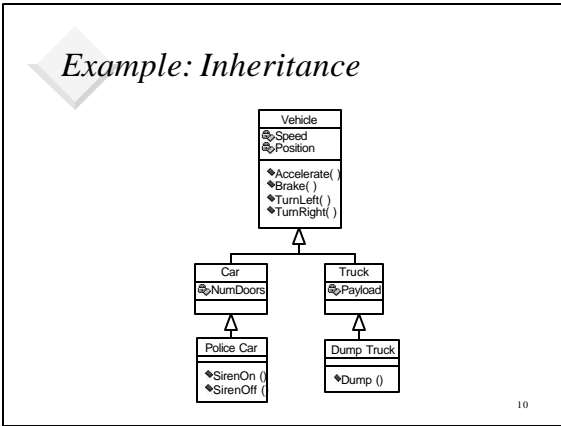
---

---

---

---

---




---

---

---

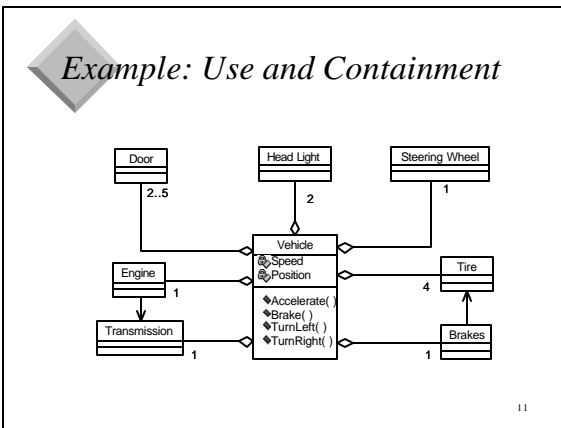
---

---

---

---

---




---

---

---

---

---

---

---

---

*Object Vars vs. C++ Pointers*

- Object variables are not like C++ references: there are no reference assignments; no NULL references
- ❖ Object variables behave more like “smart” pointers (if they existed in C++)
  - ⊙ All objects are allocated on the heap (memory pool)
  - ⊙ Using uninitialized references causes run-time error
  - ⊙ One variable assigned to another points to the same object
  - ⊙ Garbage collection - no need to reclaim memory
  - ⊙ Copy of an object is created by using *clone* method

12

---

---

---

---

---

---

---

---

### Method Parameters: Java vs. C++

- In C++ parameters can be passed by value, by reference or as de-referenced pointers
- In Java parameters are always passed by value. One should never try to change a parameter inside a function call

```
static void swap_courses( Course a, Course b)
{ // This is not going to work
  Course temp = b;
  b = a;
  a = temp;
}
```

13

---

---

---

---

---

---

---

---

### Different types of methods

- ❖ Public methods
    - accessible from anywhere
  - ❖ Private methods
    - accessible only within the class
    - normally used for internal implementation
  - ❖ Static methods
    - accessible w/o creating an instance
- ⇒ Example: `double x = Math.pow(3, 0.1); // x = 30.1`

14

---

---

---

---

---

---

---

---

### Different types of methods (cont.)

- ❖ Default (no keyword) methods
    - accessible from within the same package
  - ❖ Protected methods
    - (outside of the package) accessible only through inheritance
    - need to be overwritten in order to use
- ⇒ Example: `Object.clone()` method

15

---

---

---

---

---

---

---

---

*Constructor Method*

- ❖ Constructor
  - Creates an instance, initializes inst. variables
  - Local variables: **initialize explicitly**
- ❖ No need for destructors in Java
  - memory resources are freed up by Garbage Collector
- ❖ Object is destroyed *after* the last reference to it goes out of scope

16

---

---

---

---

---

---

---

---

*finalize() Method*

- ❖ Takes care of freeing up non-memory resources (e.g., files, sockets, descriptors)
- ❖ Called just before destroying the object
  - may never be called
  - sequence of calls is not maintained
- ❖ Use `System.RunFinalizersOnExit()` to guarantee finalizing before exit

17

---

---

---

---

---

---

---

---

*Mutator and Accessor Methods*

- ❖ Needed to change or retrieve encapsulated data from an instance
  - `FieldType getFieldname();`
  - `void setFieldName(FieldType f);`
- ❖ Classes that don't have mutators are called immutable (they cannot be changed once instantiated)

18

---

---

---

---

---

---

---

---

**Overloading**

- ❖ Method overloading
  - using the same method with a different set of arguments/parameters
  - `String s1 = new String();`
  - `String s2 = new String("Attractive");`
- ❖ Operators cannot be overloaded by a programmer.
  - Language itself has operator overloading
  - ⇒ Example: '+' as string concatenation

19

---

---

---

---

---

---

---

---

**Packages**

- ❖ Similar to libraries
  - convenient way to organize classes
- ✗ Different from libraries
  - support hierarchical naming convention
  - `freelancesoftware.bestjavautilities`
- ❖ Implemented as subdirectories
- ❖ Classed can be referred to
  - explicitly as `packageName.className`
  - implicitly using `import` statement

20

---

---

---

---

---

---

---

---

**Exercise: Create Employee class**

- ❖ Attributes:
  - name,
  - salary,
  - hireDay
- ❖ Methods:
  - `raiseSalary()`,
  - `print()`,
  - `getName()`
- ❖ Class to test class Employee
  - raise salary every year then print

21

---

---

---

---

---

---

---

---

*Useful Class Design Rules*

- ❖ Keep data private
  - Use accessor methods to access and mutator methods to change object's data.
- ❖ Explicitly initialize data
  - Don't rely on language or compiler defaults
- ❖ Combine multiple logically related variables into a class
  - *street, city, state, zip* make class *Address*

22

---

---

---

---

---

---

---

*Useful Class Design Rules (2)*

- ❖ Use standard class definition layout
  - follow visibility order
    - public features, private features, package features
  - within each group list
    - constants, constructors, methods, instance variables
- ❖ Use self-explanatory names
  - *MalingAddress, SpecialOrder, PressureGage*
  - *getPressure, setTimeOut, play, record*

23

---

---

---

---

---

---

---

*Homework Assignment*

Compile a list of differences between C++ and Java in terms of definition and implementation of a class

24

---

---

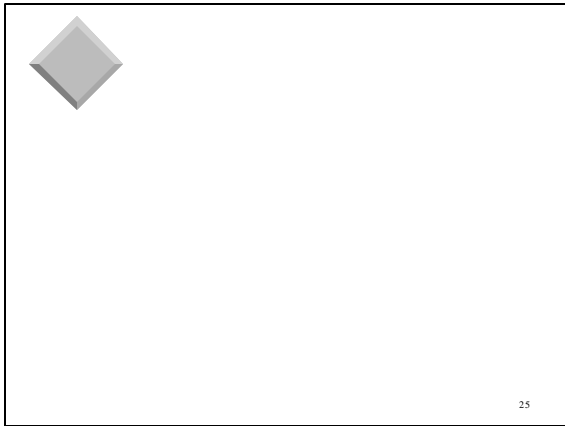
---

---

---

---

---



---

---

---

---

---

---

---