Object Oriented Thinking

Ch 10
Objectives

• Review UML class diagrams
• Understand encapsulation
• How are classes related?
• Wrapping primitives in objects
• Exploring string manipulation
UML Class Diagram

• A visual representation of a class
Fields: these represent the class attributes; usually private access.

Methods: these represent the behavior of the class; usually public access.
Properties and methods can be both private and public, but also protected and package.

- private, accessed only by defining class  
# protected, accessed by any child class in any package  
  : package, accessed by any class inside same package  
+: public, accessed by any class in any package
- Private

Within Defining Class

-: private, accessed only by defining class
#: protected, accessed by any child class in any package
Within Defining Class

Within Package Class

: package, accessed by any class inside same package
Within Defining Class

Within Any Class

+: public, accessed by any class in any package
The Customer class

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>- custId: String</td>
</tr>
<tr>
<td>- first: String</td>
</tr>
<tr>
<td>- last: String</td>
</tr>
</tbody>
</table>

+ Customer() 
+ Customer(cid: String, first: String, last: String) 
+ getCustId(): String 
+ getFirst(): String 
+ getLast(): String 
+ setCustId(cid: String): void 
+ setFirst(first: String): void 
+ setLast(last: String): void 

Property:
<access><name>: <type>
-custId: String

Method:
<access><name>( [<parameter>: <type>, ...]): <return type>
Encapsulation = state + behavior

A class encapsulates its state (fields, i.e. storage) with the behavior (methods that manipulate the state)
Implementation: how a class is coded - client does not see

Interface: what features does the class offer - client uses
private String custId;
private String first;
private String last;

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>-custId: String</td>
</tr>
<tr>
<td>-first: String</td>
</tr>
<tr>
<td>-last: String</td>
</tr>
</tbody>
</table>

Implementation: how a class is coded - client does not see

Interface: what features does the class offer - client uses
public Customer() {
    this("na","na","na");
}

public Customer(String cid,  
                String first, String last) {
    custId = cid;
    this.first = first;
    this.last = last;
}

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Customer()</td>
</tr>
</tbody>
</table>
| +Customer(cid: String,  
           first: String,  
           last: String) |

Implementation: how a class is coded - client does not see

Interface: what features does the class offer - client uses
public String getCustId() { return custId; }
public String getFirst() { return first; }
public String getLast() { return last; }

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>+getCustId(): String</td>
</tr>
<tr>
<td>+getFirst(): String</td>
</tr>
<tr>
<td>+getLast(): String</td>
</tr>
</tbody>
</table>

Implementation: how a class is coded - client does not see

Interface: what features does the class offer - client uses
Implementation: how a class is coded - client does not see

Interface: what features does the class offer - client uses
Classes can be related

1..5  *

Association
Aggregation
Composition
Generalization
Association

University \( \overset{1}{\longrightarrow} \) employs \( \overset{*}{\longrightarrow} \) Employees
Aggregation

Residence Hall

houses

Student
Composition

University has Address
Generalization

Graduate is a Student
Primitive types are backed by Objects
Boxing wraps a primitive into an object
Unboxing
unwraps an object into a primitive

int

Integer
Java offers several wrapper classes

- Float
- Double
- Character
- Boolean

There are other variations to these (Byte, Long, Short, etc)
Get to know Strings
String immutable = new String("Can't change.");
// not a common way to create a string

String immutable = "Can't change.";
// this is the more common way

char[] array = {'J', 'a', 'v', 'a'};
String language = new String(array);
// this could be convenient at times
Strings are immutable, meaning their memory cannot be changed. When a string is assigned a new literal, new memory is created for the String object.
When the string literal is the same as one used before, the string constant pool is used.

A new String object is not created, but rather the old one is reused.
The `.split` method is useful in tokenizing a string into individual tokens. The split is made on the specified delimiter.
A string can be converted to a char array

```java
char[] chars = "Java".toCharArray();

chars[0]: 'J'
chars[1]: 'a'
chars[2]: 'v'
chars[3]: 'a'
```
A string can be converted to numeric types

```java
int intVal = Integer.parseInt("195");

double dblVal = Double.parseDouble("12.34");
```
Strings can be formatted

String.format("%.2f", 123.456);
   // 123.46

System.out.printf("%.2f", 123.456);
   // 123.46