Recap: Are UML diagrams useful?

Communication
- Forward design (before coding)
  - brainstorm ideas (on whiteboard or paper)
  - draft and iterate over software design

Documentation
- Backward design (after coding)
  - obtain diagram from code

Code generation
- Automatically derive code from diagrams

Recap: Basic notation of UML class diagrams

<table>
<thead>
<tr>
<th>MyClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>- attr1 : type</td>
</tr>
<tr>
<td># attr2 : type</td>
</tr>
<tr>
<td>+ attr3 : type</td>
</tr>
<tr>
<td>~ bar(a:type) : ret_type</td>
</tr>
<tr>
<td>+ foo() : ret_type</td>
</tr>
</tbody>
</table>

Name
Attributes
- visibility
  - <visibility> <name> : <type>

Methods
- visibility
  - <visibility> <name>(<param>*) : <return type>

Visibility
- private
- package-private
- protected
+ public

Static attributes or methods are underlined

Recap: Classes, abstract classes, and interfaces

<table>
<thead>
<tr>
<th>MyClass</th>
</tr>
</thead>
</table>
| public class MyClass {
|   public void op() {
|     ...
|   }
|   public int op2() {
|     ...
|   }
| }

| MyAbstractClass |
| {abstract} |
| public abstract class MyAbstractClass {
|   public abstract void op();
|   public int op2();
| }

| MyInterface |
| <<interface>> |
| public interface MyInterface {
|   public void op();
|   public int op2();
| }
Recap:
Inheritance
Person
Student

public class Student
    extends Person{
 public Student(){
 }
 ...
}

is-a relationship

Recap:
Inheritance vs. (Aggregation vs. Composition)
Person
Student
Bank
Building

public class Student
    extends Person{
 ...
}

is-a relationship

has-a relationship

Today
More on best practices and software design
- A little refresher on polymorphism
- Live coding examples

Coding example: cs320/GetMin.java

```java
... 
LinkedList<Integer> list1 = new LinkedList<>(...);
LinkedList<Integer> list2 = new LinkedList<>(...);
Integer min1 = getMin(list1);
Integer min2 = getMin(list2);
System.out.println("Min list1: "+ min1);
System.out.println("Min list2: "+ min2);
}

private static Integer getMin(LinkedList<Integer> list) {
 sort(list);
 return list.get(0);
}

private static void sort(LinkedList<Integer> list) {
 ... // sort the list
}
```

Source code is available on the course web site.
What if we want to use ArrayLists instead?

```java
ArrayList<Integer> list1 = new LinkedList<>(...);
ArrayList<Integer> list2 = new ArrayList<>(...);
Integer min1 = getMin(list1);
Integer min2 = getMin(list2);
System.out.println("Min list1: " + min1);
System.out.println("Min list2: " + min2);
}
```

What if we want to use an ArrayList and a LinkedList?

```java
List<Integer> list1 = new LinkedList<>(...);
List<Integer> list2 = new ArrayList<>(...);
Integer min1 = getMin(list1);
Integer min2 = getMin(list2);
System.out.println("Min list1: " + min1);
System.out.println("Min list2: " + min2);
}
```

What is Polymorphism?

We can solve these problems with subtype polymorphism.
What is Polymorphism?

An object's ability to provide different behaviors.

Types of polymorphism

- Ad-hoc polymorphism (e.g., operator overloading)
  - $a + b$ ⇒ String vs. int, double, etc.

- Subtype polymorphism (e.g., method overriding)
  - Object obj = ...;
  - toString() can be overridden in subclasses
  - obj.toString(); and therefore provide a different behavior.

Coding example: cs320/PrintObject.java

```java
... String str = "Hello world!";
  Integer i = new Integer(1);
  Double d = new Double(1d);
  printString(str);
  printInteger(i);
  printDouble(d);
}
private static void printString(String str) {
  System.out.println(str.toString());
}
private static void printInteger(Integer i) {
  System.out.println(i.toString());
}
private static void printDouble(Double d) {
  System.out.println(d.toString());
}
```

Can you improve this code using subtype polymorphism?

Source code is available on the course web site.
What is Polymorphism?

An object's ability to provide different behaviors.

Types of polymorphism

- Ad-hoc polymorphism (e.g., operator overloading)
  - a + b ⇒ String vs. int, double, etc.

- Subtype polymorphism (e.g., method overriding)
  - Object obj = ...; ⇒ toString() can be overridden in subclasses and therefore provide a different behavior.

- Parametric polymorphism (e.g., Java generics)
  - class LinkedList<E> { ⇒ A LinkedList can store elements regardless of their type but still
    void add(E) {...} provides full type safety.
    E get(int index) {...}

Coding example: cs320/Poly.java, cs320/Raw.java

Generics vs. raw types

- Compare paramPoly() in cs320/Poly.java with rawTypes() in cs320/Raw.java.
- Add a String to the list in both methods (list.add(“Hello”)).
- Compile and run the code → what difference do you observe?

Poly.java raises a compile-time exception whereas Raw.java raises a runtime exception!

Inheritance: (abstract) classes and interfaces

SequentialList {abstract}

LinkedList
Inheritance: (abstract) classes and interfaces

- Iterable
- Collection
- List
- SequentialList {abstract}
- Deque
- LinkedList

List extends Iterable, Collection