CS 320

Introduction to Software Engineering
Spring 2017

March 01, 2017

Recap: Are UML diagrams useful?

Communication

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

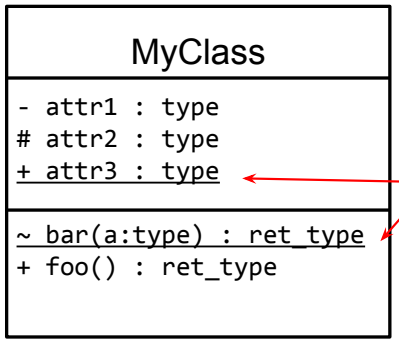
Documentation

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

Code generation

- Automatically derive code from diagrams

Recap: Basic notation of UML class diagrams



Name

Attributes

<visibility> <name> : <type>

Static attributes or methods are underlined

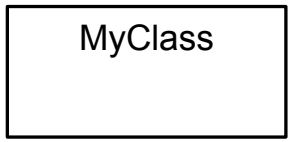
Methods

<visibility> <name>(<param>) : <return type>*
<param> := <name> : <type>

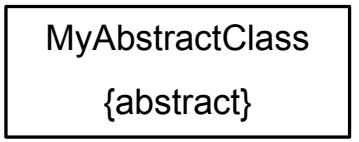
Visibility

- private
~ package-private
 # protected
+ public

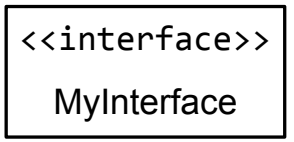
Recap: Classes, abstract classes, and interfaces




```
public class MyClass {

  public void op() {
    ...
  }

  public int op2() {
    ...
  }
}
```

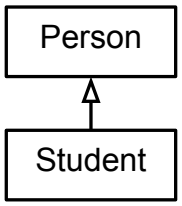



```
public abstract class
    MyAbstractClass {

  public abstract void op();


  public int op2() {
    ...
  }
}
```




```
public interface
    MyInterface {

  public void op();


  public int op2();
}
```

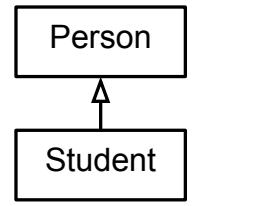
Recap:
Inheritance




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

 ...
}
```

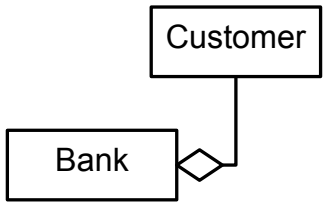
is-a relationship

Recap:
Inheritance vs. (Aggregation vs. *Composition*)

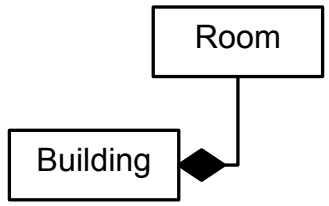



```
public class Student
    extends Person{
 public Student(){
 }

 ...
}
```

is-a relationship




```
public class Bank {
 Customer c;
 public Bank(Customer c){
  this.c = c;
 }
 ...
}
```




```
public class Building {
 Room r;
 public Building(){
  this.r = new Room();
 }
 ...
}
```

has-a relationship

Today

More on best practices and software design

- A little refresher on polymorphism
- Live coding examples

Coding example: cs320/GetMin.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.

Coding example: cs320/GetMin.java

```
  ...
  ArrayList<Integer> list1 = new LinkedList<>(...);
  ArrayList<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
}
```

What if we want to use ArrayLists instead?

Coding example: cs320/GetMin.java

```
  ...
  LinkedList<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);
}

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

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

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

Coding example: cs320/GetMin.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);
}

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

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

We can solve these problems with subtype polymorphism.

What is 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.

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.
    `obj.toString();`

Coding example: cs320/PrintObject.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.

Coding example: cs320/PrintObject.java

```
  ...
  String str = "Hello world!";
  Integer i = new Integer(1);
  Double d = new Double(1d);

  printObject(str);
  printObject(i);
  printObject(d);
}

private static void printObject(Object o) {
  System.out.println(o.toString());
}
```

printObject only relies on methods declared in the class Object.

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

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?

Source code is available on the course web site.

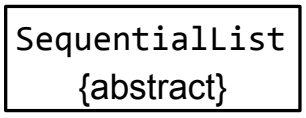
Coding example: cs320/Poly.java, cs320/Raw.java

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?

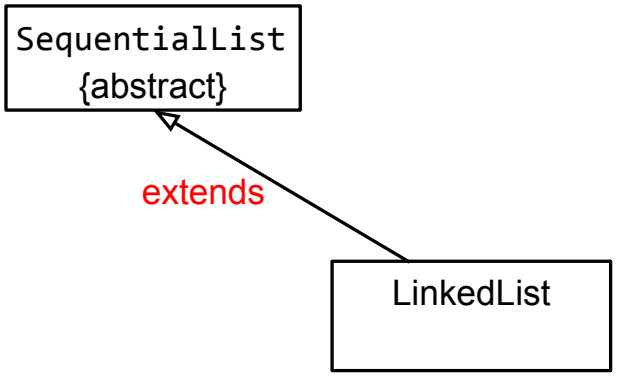
Poly.java raises a compile-time exception whereas Raw.java raises a runtime exception!

Inheritance: (abstract) classes and interfaces



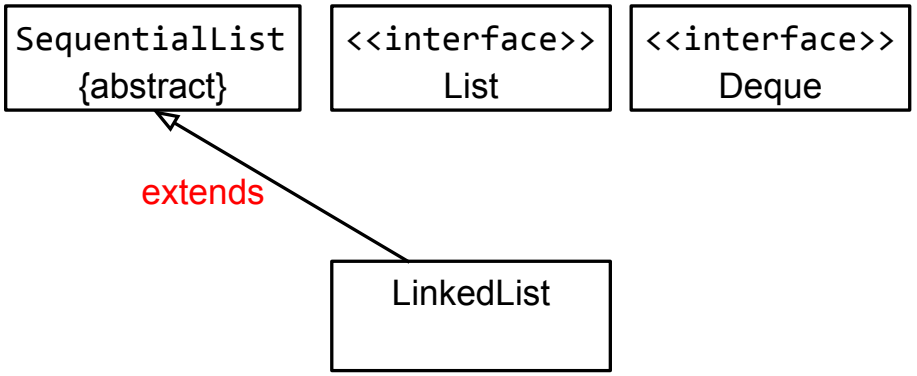
Inheritance: (abstract) classes and interfaces

LinkedList extends SequentialList



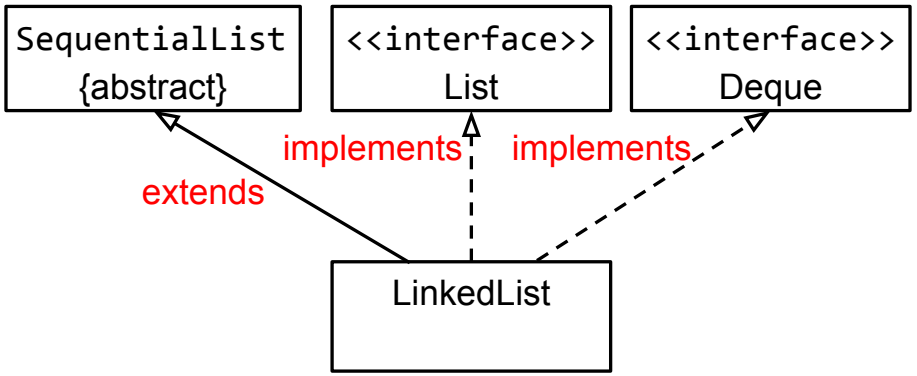
Inheritance: (abstract) classes and interfaces

LinkedList extends SequentialList

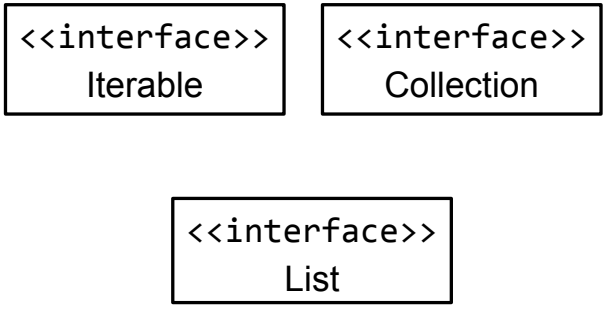


Inheritance: (abstract) classes and interfaces

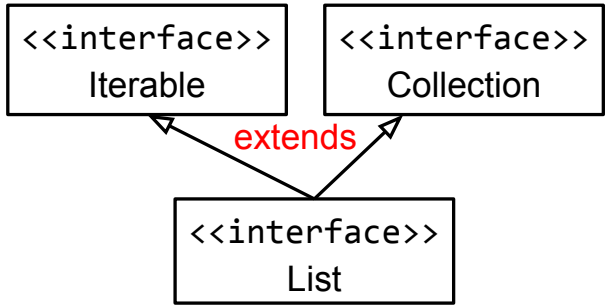
LinkedList extends SequentialList implements List, Deque



Inheritance: (abstract) classes and interfaces



Inheritance: (abstract) classes and interfaces



List extends Iterable, Collection

Inheritance: (abstract) classes and interfaces