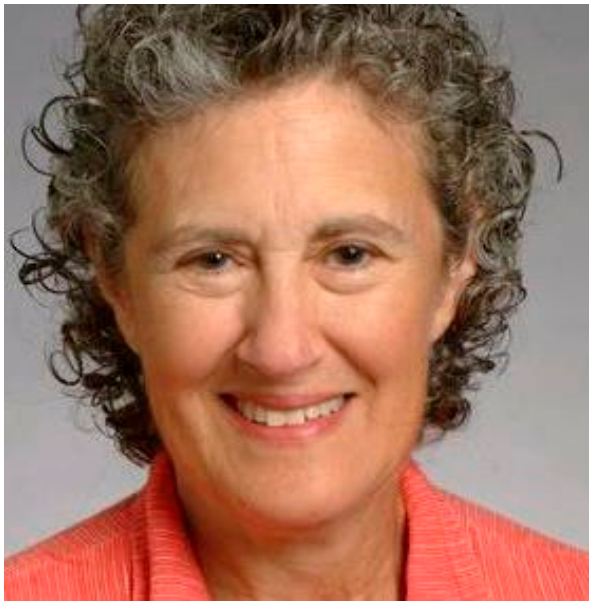
Abstract Data Types (ADT) / Interfaces

Barbara Liskov

- First woman to get a Ph.D. in Computer Science in the USA (Stanford 1968)
- Turing Award, 2008
- Inventor of Abstract Data Types



Abstract Data Types

- An approach to computer representation of abstraction
- Only the use which may be made of an abstraction is relevant
- How the abstraction is implemented is irrelevant.
- Defines a class of abstract objects which is completely characterized by the operations (functions/methods) available on those objects
- An abstract data type can be defined by defining the characterizing operations for that type

ADTs: What and Why?

- Let's consider a "Person" ADT with one "operation": greet()
    - When you want to get a person's attention, you'll address them somehow.
- There are lots of different varieties of people, each of which you'd greet differently:
    - Professor → "Hi, Professor Ives."
    - Friend → "Hey, what's up?"
    - Judge → "Hello, Your Honor."
    - Stranger → "Excuse me, hi."

ADTs: What and Why?

- Let's consider a "Person" ADT with one "operation": greet()
  - When you want to get a person's attention, you'll address them somehow.
- There are lots of different varieties of people, each of which you'd greet differently:
  - Professor → "Hi, Professor Ives."
  - Friend → "Hey, what's up?"
  - Judge → "Hello, Your Honor."
  - Stranger → "Excuse me, hi."

Each of these is a Person. BUT! None of them is *just* a Person. Every Person that you meet is more concretely something else.

ADT in Java: `interfaces`

- An interface
  - Defines an ADT in Java
  - An interface is a *class-like* construct that contains only constants and abstract methods
  - An abstract method is a method that is not implemented. Only the method signature is listed
  - A constant is a variable which value does not change during the execution of the program. They are declared static and final
  - Gives a type for an object based on what it *does*, not on how it was implemented
  - Describes a contract that objects must satisfy

Defining an interface

```
public interface InterfaceName {
   constant declarations;
   abstract method signatures;
}
```

- Example:

```
public interface Shape {
   public static final double PI = 3.14159;
   public double area();
   public double perimeter();
   public void draw();
}
```

Implementing an interface

- Define a class that will implement the interface
- The class implementing the interface must implement all the methods defined in the interface
- The class implementing an interface declares a **subtype** of the interface
- The interface is a **supertype** of the implementation class
- A class can have multiple supertypes
- An interface can have multiple subtypes

```
public class Circle implements Shape
{    (truncated for space)

  private double radius;

  public Circle(double radius){
    this.radius = radius;
  }

  @Override
  public double area() {
    return radius * radius * PI;
  }
  ...
}
```

Defined in the interface

Implementing an interface: `@Override`

- The `@Override` keyword indicates that the method implements/overrides a method defined in the interface
- Optional but very useful
- If the interface changes, methods “decorated” with `@Override` keyword will raise a compiler error. To fix the problem, make your code to adhere to the new interface

Using an interface

- Declare an object of type the interface and initialize it using the subtype constructor.
- Invoke the methods defined in the ADT on the object
- Example:

```
Shape c = new Circle(4);
c.area();
c.perimeter();
c.draw();
```

Using Abstract Data Types

1. An abstract object (an ADT is the object's type) may be operated upon by the operations which define its abstract type
2. An abstract object may be passed as a parameter to a procedure (function/method)
3. An abstract object may be assigned to a variable, but only if the variable is declared to hold objects of that type