Abstraction and Modularity

Selim Temizer
Agenda

- Abstracting Functionality
  - Black-box approach (input / output)
  - Java: Interfaces

- Modularity
  - Break down large problems into small problems

- State Machines
  - A way of describing behavior
  - ZBug
Abstraction

- Black-box approach:
  - Input: Dirty dishes
  - Output: Clean dishes
Abstraction

cleanDishes = dishwasher.wash(dirtyDishes);
cleanDishes = mom.wash(dirtyDishes);
Java - Interface

- An interface is a group of related methods with empty bodies.

- An interface specifies the inputs and outputs of our black-boxes (it is a contract).

- Dishwasher and mom are instances of classes that implement IWasher interface.
Example - Polygon Information

Define interfaces using `interface` keyword
Use interfaces like `classes`
Implement interfaces to be able to instantiate

```java
public interface IPolygon
{
    String getName();
    int getNumberOfEdges();
}

public static void information ( IPolygon p )
{
    System.out.println( p.getName() + " has " + p.getNumberOfEdges() + " edges." );
}
```
Example - Polygon Information

- Triangle class **implements** IPolygon interface
  - We need to implement all empty methods
Example - Polygon Information

- Square class implements IPolygon interface

```java
public class Square implements IPolygon
{
    public String getName()
    {
        return "Square" ;
    }

    public int getNumberOfEdges()
    {
        return 4 ;
    }
}
```
Example - Polygon Information

- Pentagon class **implements** IPolygon interface
public class PolygonInformation {
    public static void information ( IPolygon p )
    {
        System.out.println( p.getName() + " has " + p.getNumberOfEdges() + " edges." );
    }

    public static void main ( String[] args )
    {
        IPolygon myTriangle = new Triangle();
        IPolygon mySquare   = new Square();
        IPolygon myPentagon = new Pentagon();

        information( myTriangle );
        information( mySquare  );
        information( myPentagon );
    }
}

Triangle has 3 edges.
Square has 4 edges.
Pentagon has 5 edges.
Another Example

- EXCHANGE
  - exchangeDollars

- MERCHANT
  - getFood
  - getDrink

- KNOWLEDGEABLE
  - getDirections
Java - Interfaces

```java
public interface IKnowledgeable
{
    public String getDirections ( String desiredPlace ) ;
}
```

```java
public interface IExchange
{
    public float exchangeDollars ( float dollars ) ;
}
```

```java
public interface IMerchant
{
    public void getFood ( float shekels ) ;
    public void getDrink ( float shekels ) ;
}
```
Use Interfaces like Classes

```java
public static void act ( IKnowledgeable k, IExchange e, IMerchant m )
{
    String directionsToExchange = k.getDirections( "Exchange Place" );
    String directionsToFoodShop = k.getDirections( "Food Place" );

    float receivedShekels = e.exchangeDollars( 50.0 );
    m.getFood( receivedShekels / 2 );
    m.getDrink( receivedShekels / 2 );
}
```
public class StudentInfo implements IKnowledgeable
{
    public String getDirections ( String desiredPlace )
    {
        if ( desiredPlace.equals( "Exchange Place" ) ) { return "Exchange Place is downstairs, on the left" ; }
        if ( desiredPlace.equals( "Food Place" ) ) { return "Food Place is upstairs" ; }
    }
}

class StudentExchange implements IExchange
{
    public float exchangeDollars ( float dollars )
    {
        return ( dollars * 3.45480425f ) ;
    }
}

class StudentSell implements IMerchant
{
    public void getFood ( float shekels )
    {
        System.out.println( "Here is your food" ) ;
    }

    public void getDrink( float shekels )
    {
        System.out.println( "Here is your drink" ) ;
    }
}
public class StudentInfoExchange implements IKnowledgeable, IExchange {

    public String getDirections ( String desiredPlace ) {
        if ( desiredPlace.equals( "Exchange Place" ) ) { return "I can exchange money" ; }
        if ( desiredPlace.equals( "Food Place" ) ) { return "Food Place is upstairs" ; }
    }

    public float exchangeDollars ( float dollars ) {
        return ( dollars * 3.45480425f ) ;
    }
}
public class StudentInfoExchangeSell implements IKnowledgeable, IExchange, IMerchant {

    public String getDirections ( String desiredPlace ) {
        if ( desiredPlace.equals( "Exchange Place" ) ) { return "I can exchange money"; }
        if ( desiredPlace.equals( "Food Place" ) ) { return "I can sell you food"; }
    }
}

public float exchangeDollars ( float dollars ) {
    return ( dollars * 3.45480425f );
}

public void getFood ( float shekels ) {
    System.out.println( "Here is your food" );
}

public void getDrink( float shekels ) {
    System.out.println( "Here is your drink" );
}
public class Selim
{

    public static void act ( IKnowledgeable k , IExchange e , IMerchant m )
    {
        String directionsToExchange = k.getDirections( "Exchange Place" );
        String directionsToFoodShop = k.getDirections( "Food Place" );

        float receivedShekels = e.exchangeDollars( 50.0 );

        m.getFood ( receivedShekels / 2 );
        m.getDrink( receivedShekels / 2 );
    }

    //-------------

    public static void main ( String[] args )
    {
        StudentInfo Liel      = new StudentInfo ();
        StudentExchange Sadek  = new StudentExchange ();
        StudentSell Talya     = new StudentSell ();
        StudentInfoExchange Mustafa = new StudentInfoExchange ();
        StudentInfoExchangeSell Yohanan = new StudentInfoExchangeSell();

        act( Liel , Sadek , Talya ); // OK
        act( Mustafa, Mustafa, Talya ); // OK
        act( Yohanan, Yohanan, Yohanan ); // OK
    }
}
Interfaces - Summary

- Specify the **functions** we need (input and output)
- Declared by the **interface** keyword
- **Used** like classes
- **Cannot be instantiated**
- We need to instantiate classes that **implement** interfaces
Modularity

- General concept

- Break a large system into smaller, independent **modules**
Modularity - Software

- Break **large programs** into smaller, independent **classes**
- Break **large methods** into smaller **methods**
Modularity - Software

Before

```java
public void act ()
{
    // count = number of turns before move()
    for ( int i = 0 ; i < count ; i++ )
    {
        turn() ;
    }
    move() ;
}
```

After

```java
public void act ()
{
    // count = number of turns before move()
    turnNTimes( count ) ;
    move() ;
}
```
Why Modules?

• Reuse code
  ○ Example: we can call `turnNTimes` again

• Divide work for multiple programmers
  ○ Define interfaces for each module

• Test / Debug modules independently
  ○ Easy to find bugs in your Java program

• Easy modification and maintenance
  ○ Modify / fix only a small amount of code
State Machine

● What is a state machine?
  ○ A simple way to describe the behavior of a system

● What is a state?
  ○ All the information you need to know in order to pick your next action

● How to build a state machine?
  ○ Design your states
  ○ Choose your start state
  ○ Specify how/when to move from a state to another state
Wait for money

Put enough money

Wait for choice

Select drink

Give drink
How does ZBug behave?
ZBug

- **State:**
  - int zSize;
  - int part;
  - int numberOfStepsTaken;

- **Initial values**
  - zSize = given by user
  - part = 1
  - numberOfStepsTaken = 0