

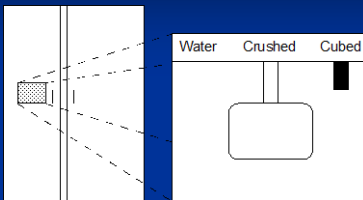
Intro to Design & Specification Using Java/UML

- For the use of CP SC 872 students
- Slides of Bruce W. Weide, CSE, The Ohio State University
- Do not copy/reproduce without permission of Weide

Systems Thinking

- **System** is:
 - any part of anything that you want to think about as an indivisible unit
- **Interface** is:
 - a description of the “boundary” between a system and everything else, that also explains how to think about that system as a unit
- **Subsystem/component** is:
 - a system that is used inside, i.e., as a part of, another system (a relative notion!)

Example: Ice/Water Dispenser



Select water, crushed ice, or cubed ice.
Place a glass against the pad and push.

People's Roles wrt Systems

- **Client** is:
 - a person (or a “role” played by some agent) viewing a system “from the outside” as an indivisible unit
- **Implementer** is:
 - a person (or a “role” played by some agent) viewing a system “from the inside” as an assembly of subsystems/components

Interfaces: Describing Behavior

- **Information hiding** is:
 - a technique where you intentionally leave out information that is merely an “internal implementation detail” of the system
- **Abstraction** is:
 - a complementary technique where you create a valid “cover story” to mask the effects of hiding (presumably important) information about “internal implementation details”

Main

- **visible to a client** (points to `main`)
 - **a Java “system”** (points to `Main`)
 - **a class method** (points to `main`)
 - **returns no value (a.k.a. a “procedure”)** (points to `void`)
 - **Java type for text** (points to `String`)
- ```
{
 public static void main (String args[])
 {
 System.out.println ("Hello, world!");
 }
}
```

- Note: “System.out” in this code is really a subsystem; don’t let the name confuse you!

## Another Example: Demo01

- *declaration of the object/instance hw*
- *creation of hw by instantiating HelloWorld\_1 (calling its constructor)*
- *method call of outputHelloWorld on receiver object hw*

```

public class HelloWorld {
 public static void main (String args[]) {
 IHelloWorld hw = new HelloWorld_1 ();
 hw.outputHelloWorld ();
 }
}

```

## Make a Java "Interface" to Java

- Interface or subsystem an *instance method* code:
- *behavioral specification of the method, a.k.a. its contract*

```

public interface IHelloWorld {
 public void outputHelloWorld ();
 /*
 * ensures
 * ["Hello, world!" appears on
 * System.out]
 */
}

```

## Implement an Interface

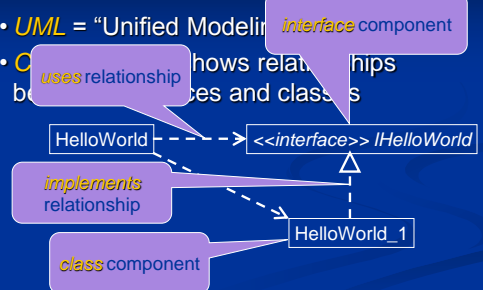
```

public class HelloWorld_1
 implements IHelloWorld {
 public void outputHelloWorld () {
 System.out.println ("Hello, world!");
 }
}

```

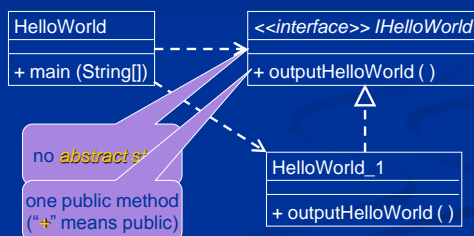
## UML Class Diagram

- *UML* = "Unified Modeling Language"
- *UML* shows relationships between objects and classes
- *uses relationship*
- *implements relationship*
- *interface component*
- *class component*



## Enhanced Class Diagram

- Class diagram can contain more info:



## Running a Java Program

- The class *HelloWorld* must be in a file called *HelloWorld.java* (similarly for an interface, e.g., *IHelloWorld.java*)
- To compile the class with the main program *and* all components it depends on:
 

```
javac HelloWorld.java
```
- To run the program ("execute" it):
 

```
java HelloWorld
```

## What Happens If...

- You change the interface *IHelloWorld* so:
  - the name of the interface is *IHW*?
  - the method is called *printHelloWorld*?
  - the specification says the output is "Hello, Barack!"?

## What Happens If...

- You change the class *HelloWorld\_1* so:
  - it doesn't claim it implements *IHelloWorld*?
  - the method is called *printHelloWorld*?
  - the output string is "Hello, Mr. Obama!"?

## What Happens If...

- You change the class *HelloWorld* so:
  - *main* has no arguments?
  - the parameter to *main* is called *theArgs*?
  - the type of *hw* is *HelloWorld\_1*?
  - the constructor after **new** is *IHelloWorld ( )*?