CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce
343 Davis Hall
alphonce@buffalo.edu

Office hours:
Thursday 12:00 PM – 2:00 PM
Friday 8:30 AM – 10:30 AM
OR request appointment via e-mail
Turn off and put away electronics:

- cell phones
- pagers
- laptops
- tablets
- etc.
Where we’ve been
- Exam return
- Interfaces (intro)

Today
- Interfaces
- event-driven programming
- control structures

Where we’re heading
- collections
- search
EXAM RESULTS
MAX 100 (x13 = 4.2%)
MEDIAN 76
MEAN 74.3
MIN 8

CSE115 Exam 1 Grade Distribution

Well done!
Let’s talk
Keep working hard

Well done!
Let’s talk
Keep working hard
I have unclaimed exams with me.

Pick up at end of class.

Grading questions? Come to office hours.
channel 1
Which of the following is not represented in an object diagram?

A. method
B. object
C. variable
D. reference
Which of the following is not represented in an object diagram?

A. method
B. object
C. variable
D. reference

Convince your neighbor your answer is correct.
Which of the following is not represented in an object diagram?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A.</td>
<td>method</td>
</tr>
<tr>
<td>B.</td>
<td>object</td>
</tr>
<tr>
<td>C.</td>
<td>variable</td>
</tr>
<tr>
<td>D.</td>
<td>reference</td>
</tr>
</tbody>
</table>
INTERFACES
header + body

header
  access control modifier
  keyword ‘interface’
  name (generally an adjective, following class name conventions, but prefixed with an upper-case ‘I’)

body
  method specifications (method headers followed by ‘;’, also called method declarations, as opposed to method definition)
  a few other things are permitted in interfaces (e.g. Java 8 now allows “default methods”) we won’t worry about these right now.
1) Example from Java’s libraries (one detail omitted)

```java
public interface ActionListener {
    public void actionPerformed(ActionEvent e);
}
```

2) Example from Java’s libraries (one detail omitted)

```java
public interface MenuKeyListener {
    void menuKeyTyped(MenuKeyEvent e);
    void menuKeyPressed(MenuKeyEvent e);
    void menuKeyReleased(MenuKeyEvent e);
}
```
While classes can be instantiated, interfaces cannot be instantiated.

Why is this?
REALIZATION
Realization

a relationship between a

class

and an

interface
An interface contains *method specifications*, rather than full method definitions.
public interface ActionListener {
    public void actionPerformed(ActionEvent e);
}

public interface BinaryOperation {
    public int perform(int arg);
}
A class can *implement* an interface:

```java
public class EventHandler implements ActionListener {
    ...
}
```
A class which implements an interface is obligated to provide full definitions of all the methods specified in the interface.
A class can implement an interface:

```java
class EventHandler implements ActionListener {
    ...
    @Override
    public void actionPerformed(ActionEvent e) {
        ...
    }
    ...
}
```
public class EventHandler implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked");
    }
}
An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.

[...]

When overriding a method, you might want to use the @Override annotation that instructs the compiler that you intend to override a method in the superclass. If, for some reason, the compiler detects that the method does not exist in one of the superclasses, then it will generate an error.

https://docs.oracle.com/javase/tutorial/java/IandI/override.html
When you define a class, you are defining a type.

When you define an interface, you are also defining a type.

A class which implements an interface is a SUBTYPE of the interface type.

an instance of the class belongs to both types
If a variable is declared to be of an interface type (e.g. IType), it can be assigned an instance of any subtype class (e.g. CType):

```java
public class C1 implements IType {...}
public class C2 implements IType {...}

IType var;
var = new C1 (); // subtype of IType
var = new C2 (); // subtype of IType
```
The **declared type** of a variable, not the actual type of the object the variable refers to, determines **WHICH methods** can be called on the object.

The **actual type** of the object on which a method is called, rather than the declared type of the variable, determines **the behavior** (the code executed).

We’ll have more to say about this when we discuss the inheritance relationship.
java.awt.event.ActionListener
ActionListener objects can serve as event handlers for JButtons.

An ActionListener object must be associated with a JButton to play this role:

```java
JButton b = new JButton("Click me!");
ActionListener e = new EventHandler();
b.addActionListener(e);
```
A JButton is a component which can react to mouse clicks.
Clicks on buttons, mouse movements, etc. are all considered events.

A program can react to events by setting up event handlers.

An event handler defines what should happen when a particular event occurs.
The component which gives rise to an event is decoupled from the part of the code that handles the event.

This is called the observer pattern.

General form:

http://www.oodesign.com/observer-pattern.html
Observer pattern in Java

An observer is called a listener in Java

Button clicks are “ActionEvents”.

Handlers for ActionEvents are ActionListener.

An event-generator can have many listeners.

Use “addActionListener” method to register a listener with a component.