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
Graphics intro

Today
exam return / results
Interfaces

Where we’re heading
event-driven programming
collections
control structures
primitives
search
EXAM RESULTS
EXAM RESULTS (FA 15)

MAX \hspace{1em} 100 (x19 = 3.4\%)
MEDIAN \hspace{1em} 68
MEAN \hspace{1em} 68.4
MIN \hspace{1em} 20
MAX      100 (x13 = 4.2%)
MEDIAN   76
MEAN     74.3
MIN      8
MAX 100 (x13 = 4.2%)
MEDIAN 76
MEAN 74.3
MIN 8
Come down to pick up your exam when your name is called.

If you arrived late, wait until the end.

Please keep noise level down so fellow students can hear their name being called.
INTERFACES
SYNTAX: form of an interface

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 is a relationship between a class and an interface.

An interface contains method specifications, rather than full method definitions.
A class can implement an interface:

```java
public class EventListener 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
public 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");
    }
}
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 C1Type implements IType {...}
public class C2Type implements IType {...}

IType var;
var = new C1Type(); // subtype of IType
var = new C2Type(); // subtype of IType
```
Method restrictions

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.
public class EventHandler implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked");
    }
}

REMEMBER THIS CLASS
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);
```