Abstract Classes and Interfaces

By the end of this video you will be able to…

- Use the keyword abstract
- Compare “inheritance of implementation” and “inheritance of interface”
- Decide between Abstract Classes and Interfaces

Person – Campus Accounts

- Add method “monthlyStatement”
- “Person” objects no longer make sense

Person – Campus Accounts

- Add method “monthlyStatement”
- “Person” objects no longer make sense

How do we:

1. Force subclasses to have this method
2. Stop having actual Person objects
3. Keep having Person references
4. Retain common Person code

Person – Campus Accounts

- Add method “monthlyStatement”
- “Person” objects no longer make sense

How do we:

1. Force subclasses to have this method
2. Stop having actual Person objects
3. Keep having Person references
4. Retain common Person code

Abstract classes!

Abstract

- Can make any class abstract with keyword:

```
public abstract class Person {
```

- Class **must** be abstract if any methods are:

```
public abstract void monthlyStatement() {
```

Implementation vs. Interface

Abstract classes offer inheritance of both!

- **Implementation:** instance variables and methods which define common behavior
- **Interface:** method signatures which define required behaviors

Implementation vs. Interface

- **Implementation:** instance variables and methods which define common behavior
- **Interface:** method signatures which define required behaviors

What if we just want the Interface?

Person – Campus Accounts

- Add method “monthlyStatement”
- “Person” objects no longer make sense

How do we:

1. Force subclasses to have this method
2. Stop having actual Person objects
3. Keep having Person references
4. ~~Retain common Person code~~

Then use an Interface!

Interfaces

- Interfaces only define required methods
- Classes can inherit from multiple Interfaces

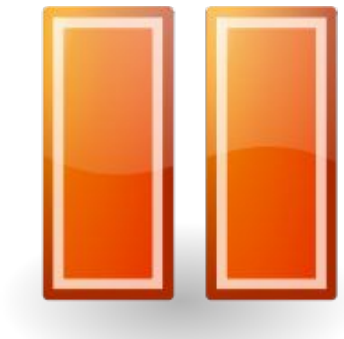
```
// Defined in java.lang.Comparable
package java.lang;

public interface Comparable<E> {
 // Compare this object's name to o's name
 // Return < 0, 0, > 0 if this object compares
 //   less than, equal to, greater than o.
  public abstract int compareTo( E o );
}
```

```
// Defined in java.lang.Comparable
package java.lang;

public interface Comparable<E> {
 // Compare this object's name to o's name
 // Return < 0, 0, > 0 if this object compares
 //   less than, equal to, greater than o.
  public abstract int compareTo( E o );
}
```

Why have this interface??



```
// Defined in java.lang.Comparable
package java.lang;

public interface Comparable<E> {
 // Compare this object's name to o's name
 // Return < 0, 0, > 0 if this object compares
 //   less than, equal to, greater than o.
  public abstract int compareTo( E o );
}
```

```
public class Person implements Comparable<Person> {
  private String name;
  ...

  @Override
  public int compareTo( Person o ) {
    return this.getName().compareTo( o.getName() );
  }
```

Abstract class or Interface?

- If you just want to define a required method:

  Interface

- If you want to define potentially required methods AND common behavior:

  Abstract class