**Administrative**

**Today’s session**

Annotation types

Interface

Interface – abstract class comparison

**Session Topics**

**Annotation types**

● An **annotation type** is an instruction to the Java compiler.

● An annotation type has syntax:

@<annotation-type>

● Package java.lang includes these annotation types:

|  |  |  |
| --- | --- | --- |
| Annotation type | Purpose | |
| @Override | | Informs the compiler that the class member is meant to override a class member declared in a superclass. |
| @SuppressWarnings(“<w>”)  OR  @SuppressWarnings({“<w-1”, “w-2”, …, “w-n”}) | | Informs the compiler to suppress the specified warning(s). The list of warnings that may be used is not consistent across development tools. Here is the list of valid [[Eclipse (Neon) warnings](http://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Ftasks%2Ftask-suppress_warnings.htm)](http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Ftasks%2Ftask-suppress_warnings.htm). |
| @Deprecated | | Informs the compiler that the marked class member is deprecated and should no longer be used. |

● These annotation types appear just before the class member they apply to.

**Interface**

● An **interface** is an extreme form of an abstract class.

● Like an abstract class, an interface forces (concrete) subclasses to implement its methods.

● Like an abstract class, an interface is a property of a class that defines what methods the class should include.

● Unlike an abstract class, an interface is a type.

● Unlike an abstract class, an interface is often applied across unrelated types of classes.

● An interface contains only:

✓ Constants

✓ Abstract methods

● An interface contains no:

✓ Instance variables

✓ Concrete methods

● An interface is declared with keyword **interface**.

● An interface has syntax:

public interface <interface-name>

{

public static final <constant-1> = <constant-value-1>;

public static final <constant-2> = <constant-value-2>;

…

public static final <constant-n> = <constant-value-n>;

public abstract <return-type> method-1(…);

public abstract <return-type> method-2(…);

…

public abstract <return-type> method-n(…);

}

● All members of an interface are public.

● All interface methods are public and abstract (abstract and static are mutually exclusive).

● All interface fields are public, static, and final (constants).

● Modifiers on members are optional.

● Interface example:

public interface CommonMembers

{

public static final float APP\_VERSION = 2.0f;

public abstract String classTypeCoded();

public abstract int hashCodeScaled(int scaleValue);

}

● A class may:

✓ Extend *one* class, abstract or otherwise.

✓ Implement *multiple* interfaces.

● To use an interface in a concrete class:

✓ Add **implements <interface-name>** to the concrete class declaration.

✓ Implement *all* of the interface methods (convert them from abstract to concrete). If a class does not implement all interface methods, it must be declared an abstract.

● Implements example:

public class GreatClass implements CommonMembers

{

…

@Override

public String classTypeCoded()

{

return this.getClass().getSimpleName().toUpperCase().substring(0,4);

}

@Override

public int hashCodeScaled(int scaleValue)

{

return this.hashCode() % scaleValue;

}

…

}

● To use an interface in an abstract class:

✓ Add **implements <interface-name>** to the abstract class declaration.

✓ Implement *any* ofthe interface methods (convert them from abstract to concrete). Any unimplemented methods are considered abstract.

● Here are some interface types provided by Java:

|  |  |
| --- | --- |
| Interface | Purpose |
| CharSequence | A readable sequence of char values. Methods that must be defined include **charAt**, **chars**, **codePoints**, **length**, **subsequence**, and **toString**. |
| Cloneable | Enable a class to be cloneable. Method **clone** must be defined to call super.clone. |
| Comparable<T> | Compare two objects of type T. Method **compareTo** must be defined to return a negative integer, zero, or a positive integer after comparing the objects. |
| List | An ordered collection. Methods that must be defined include **add**, **clear**, **contains**, **equals**, **hashCode**, **isEmpty**, **remove**, and  **size**. |
| Set | A collection that contains no duplicate elements. Methods that must be defined include **add**, **clear**, **contains**, **equals**, **hashCode**, **isEmpty**, **remove**, and  **size**. |

● If multiple interfaces are implemented in a class, an inconsistency will occur if:

✓ Two or more interfaces have the **same defined constant** but contain different values.

✓ Two or more interfaces have the **same signature** but different return types.

● Since an interface is a data type:

✓ It may be used as the data type for a method parameter. This means that the method can handle any object that implements the interface.

✓ It may be used as the return data type for a value method. This means that the returned value must be cast to any object that implements the interface.

● See **Interface type** sample application on Blackboard.

**Interface – abstract class comparison**

|  |  |
| --- | --- |
| Interface | Abstract class |
| An interface is a type. | An abstract class is a class. |
| All of its methods must be abstract. | Any of its methods may be abstract. |
| All of its members must be public. | Any of its members may be public. |
| An interface may not be instantiated. | An abstract class may not be instantiated. |
| A class may implement multiple interfaces. | A class may extend one abstract class. |
| No implementation code is provided so the implementing class must provide all of it. | Some implementation code may be provided so the extending class inherits it. |
| An interface is used to add generic functionality to a class. | An abstract class is used to provide a code framework for its subclasses. |