Chapter 11 Abstract Classes and Interfaces

Superclasses and Subclasses

<table>
<thead>
<tr>
<th>GeometricObject</th>
<th>GeometricObject1.java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circles</td>
<td>Circle4.java</td>
</tr>
<tr>
<td>Rectangles</td>
<td>Rectangle1.java</td>
</tr>
</tbody>
</table>

Abstract Classes and Abstract Methods

<table>
<thead>
<tr>
<th>GeometricObject</th>
<th>GeometricObject1.java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circles</td>
<td>Circle4.java</td>
</tr>
<tr>
<td>Rectangles</td>
<td>Rectangle1.java</td>
</tr>
</tbody>
</table>

abstract method in abstract class
- If a class contains abstract methods, it must be declared abstract
- An abstract method cannot be contained in a nonabstract class
- If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be declared abstract
- In other words, in a nonabstract subclass extended from an abstract class, all the abstract methods must be implemented, even if they are not used in the subclass

Instance cannot be created from abstract class
- An abstract class cannot be instantiated using the new operator
- You can still define its constructors, which are invoked in the constructors of its subclasses
- For instance, the constructors of GeometricObject are invoked in the Circle class and the Rectangle class.

abstract class without abstract method
- A class that contains abstract methods must be abstract
- It is possible to declare an abstract class that contains no abstract methods
- In this case, you cannot create instances of the class using the new operator - this class can be used as a base class for defining a new subclass
superclass of abstract class may be concrete

- A subclass can be abstract even if its superclass is concrete
- For example, the `Object` class is concrete, but its subclasses, such as `GeometricObject`, may be abstract.

concrete method overridden to be abstract

- A subclass can override a method from its superclass to declare it abstract
- This is rare, but useful when the implementation of the method in the superclass becomes invalid in the subclass
- In this case, the subclass must be declared abstract.

abstract class as type

- You cannot create an instance from an abstract class using the `new` operator, but an abstract class can be used as a data type

```java
GeometricObject obj = new Circle(10);
GeometricObject[] geo = new GeometricObject[10];
```

Review questions

- Which of the following declares an abstract method in an abstract Java class?
  
  A. `public abstract method();`
  B. `public abstract void method();`
  C. `public void abstract Method();`
  D. `public void method() {}`
  E. `public abstract void method() {}`

Review questions

Which of the following statements regarding abstract methods are true?

A. An abstract class can have instances created using the constructor of the abstract class.
B. An abstract class can be extended.
C. A subclass of a non-abstract superclass can be abstract.
D. A subclass can override a concrete method in a superclass to declare it abstract.
E. An abstract class can be used as a data type.

Review questions

Suppose A is an abstract class, B is a concrete subclass of A, and both A and B have a default constructor. Which of the following is correct?

A. `A a = new A();`
B. `A a = new B();`
C. `B b = new A();`
D. `B b = new B();`
What is an interface?

An interface is a class-like construct that contains only constants and abstract methods.

In many ways, an interface is similar to an abstract class, but the intent of an interface is to specify behavior for objects.

For example, we can specify that the objects are comparable, edible, cloneable using appropriate interfaces such as `Comparable`, `Edible`, and `Cloneable`.

A class that implements an interface need to implement all the abstract methods.

For example, we can define `Orange` and `Chicken` classes that implement `Edible` interface.

How is an interface useful?

Interfaces are useful for several reasons:

1. **Abstraction:** Interfaces provide a way to abstract away the details of how a task is performed, allowing for more generic and reusable code.
2. **Polymorphism:** Interfaces enable polymorphism, allowing for more flexible and dynamic programming.
3. **Type Safety:** Interfaces ensure that a class that claims to implement an interface actually provides the required methods and constants.

How do you define an interface?

Define an Interface

```java
public interface InterfaceName {
    constant declarations;
    method signatures;
}
```

```java
public interface Edible {
    /** Describe how to eat */
    public abstract String howToEat();
}
```

```java
interface Edible {
    public abstract String howToEat();
}
```

```java
Edible Edible.java
TestEdible.java
```

Omitting Modifiers in Interfaces

- All data fields are public final static (constants) in an interface.
- All methods are public abstract in an interface.

```java
public interface T1 {
    public static final int K = 1;
    public abstract void p();
}
```

```java
public interface T1 {
    int K = 1;
    void p();
}
```

Interface is a Special Class

- Like an abstract class, you cannot create an instance from an interface using the new operator.
- You can create an instance from a class that implements an interface.
- You can use an interface as a data type for a variable, as the result of casting, and so on.

The Comparable Interface

```java
// This interface is defined in
// java.lang package
package java.lang;

public interface Comparable {  
    public int compareTo(Object o);
}
```
Many classes (e.g., String and Date) in the Java library implement Comparable to define a natural order for the objects.

```java
public class String extends Object implements Comparable {
    // class body omitted
}
```

```java
public class Date extends Object implements Comparable {
    // class body omitted
}
```

Declaring Classes to Implement Comparable

```java
ComparableRectangle rectangle1 = new ComparableRectangle(4, 5);
ComparableRectangle rectangle2 = new ComparableRectangle(3, 6);
System.out.println(Max.max(rectangle1, rectangle2));
```

Card class implements Comparable!

Review questions

Which of the following is a correct interface?

A. interface A { void print() { }; }
B. abstract interface A { print(); }
C. abstract interface A { abstract void print() { } }
D. interface A { void print(); }

Using Interface for Event Programming

Event programming

- the flow of the program is determined by user actions (mouse clicks, key presses) or messages from other programs.

Components

- Event sources: user interface components or other sources that generate the events
- Events: user actions or other events
- Event listener: reactions on events

Basic steps

- Define event listener - implements an interface called ActionListener which contains a method actionPerformed() for processing the event
- Register event listener with event sources

Let's see how we can use GUI buttons!

It's all about the interfaces

HandleEvent.java

Generic max Method

Let's write a generic method that sorts an array of comparable objects

```java
GenericSort.java
```
Interfaces vs. Abstract Classes

- In an interface, the data must be constants; an abstract class can have all types of data.
- Each method in an interface has only a signature without implementation; an abstract class can have concrete methods.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Constructors</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract class</td>
<td>No restrictions</td>
<td>Constructors are invoked by subclassest through constructor chaining. An abstract class cannot be instantiated using the new operator.</td>
</tr>
<tr>
<td>Interface</td>
<td>All variables must be public static final</td>
<td>No constructors. An interface cannot be instantiated using the new operator. All methods must be public abstract instance methods</td>
</tr>
</tbody>
</table>

Interfaces vs. Abstract Classes, cont.

- All classes share a single root, the Object class, but there is no single root for interfaces.

Practice questions

- Abstract classes
  - 11.1, 11.4
- Interface
  - 11.4, 11.7

Caution: conflicting interfaces

- A class may implement two interfaces with conflict information (e.g., two same constants with different values or two methods with same signature but different return type).
- This type of errors will be detected by the compiler.