Abdul Haseeb

Interfaces in Java

Introduction

Achieve complete abstraction

- Specify what a class must do, but not how it does it.

Lack instance variables

May contain one or more abstract methods

A class **implements** one or more interfaces (Multiple Inhertitance)

An interface may extend another interface

Syntax

```
access interface name {
    return-type method-name1(parameter-list);
    return-type method-name2(parameter-list);

    type final-varname1 = value;
    type final-varname2 = value;
    //...
    return-type method-nameN(parameter-list);
    type final-varnameN = value;
}
```

Methods and Variables in Interface

Variables are final and static, means implementing class can’t change them

All methods and variables are implicitly public

Interface Demo

```
interface Callback {
  void callback(int param);
}
```

Implements

A class can use interface by implementing it:

- class circle **implements** shape{}
- Override the area method by making it public

A class will work normally even after implementing interfaces

Implementing an Interface

```
class Client implements Callback {
  // Implement Callback's interface
  public void callback(int p) {

    System.out.println("callback called with " + p);
  }
}
```

Normal for classes to have their own methods

```
class Client implements Callback {
  // Implement Callback's interface
  public void callback(int p) {
    System.out.println("callback called with " + p);
  }

  void nonIfaceMeth() {
    System.out.println("Classes that implement interfaces " +
                       "may also define other members, too.");
  }
}
```

Creating an Interface reference and assigning it object of a class

- Can only access those methods, which were declared in interface, because it has no knowledge of client methods!

```
class TestIface {
  public static void main(String args[]) {
    Callback c = new Client();
    c.callback(42);
  }
}
```

Call to a method is determined at run-time

```
// Another implementation of Callback.
class AnotherClient implements Callback {
  // Implement Callback's interface
  public void callback(int p) {
    System.out.println("Another version of callback");
    System.out.println("p squared is " + (p*p));
  }
}
```

Now, try the following class:

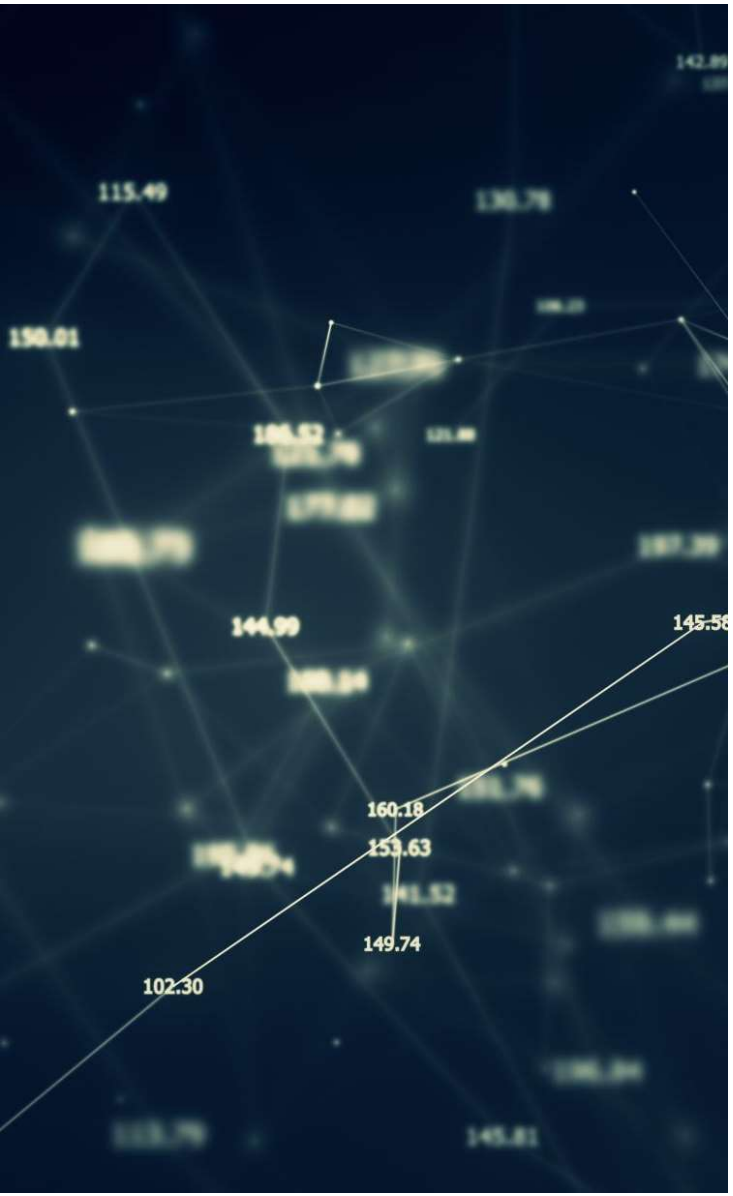
```
class TestIface2 {
  public static void main(String args[]) {
    Callback c = new Client();
    AnotherClient ob = new AnotherClient();

    c.callback(42);

    c = ob; // c now refers to AnotherClient object
    c.callback(42);
  }
}
```

Partial Implementations

- If any class implements an interface:
  - It must override or provide complete implementation of all the abstract methods of interface
  - Otherwise make that class as abstract



Nested Interface

- An Interface can be declared as a member of class or another interface
  - Nested or Member interface
- When using nested interface:
  - Qualify it with the class name or interface name, which it is a part of

```
// A nested interface example.

// This class contains a member interface.
class A {
  // this is a nested interface
  public interface NestedIF {
    boolean isNotNegative(int x);
  }
}

// B implements the nested interface.
class B implements A.NestedIF {
  public boolean isNotNegative(int x) {
    return x < 0 ? false: true;
  }
}

class NestedIFDemo {
  public static void main(String args[]) {

    // use a nested interface reference
    A.NestedIF nif = new B();

    if(nif.isNotNegative(10))
      System.out.println("10 is not negative");
    if(nif.isNotNegative(-12))
      System.out.println("this won't be displayed");
  }
}
```

Assignment

- Understand the stack example from book using Interfaces

Variables in Interface

- If an interface has no any methods, but only the variables:
  - If class implements it:
    - All those variables are available as constants in Class scope

Implementing an Automated Decision Maker

```
import java.util.Random;

interface SharedConstants {
   int NO = 0;
   int YES = 1;
   int MAYBE = 2;
   int LATER = 3;
   int SOON = 4;
   int NEVER = 5;
}
```

```
class Question implements SharedConstants {
  Random rand = new Random();
  int ask() {
    int prob = (int) (100 * rand.nextDouble());
    if (prob < 30)
      return NO;              // 30%
    else if (prob < 60)
      return YES;             // 30%
    else if (prob < 75)
      return LATER;           // 15%
    else if (prob < 98)
      return SOON;            // 13%

    else
      return NEVER;           // 2%
  }
}
```

```
class AskMe implements SharedConstants {
  static void answer(int result) {
    switch(result) {
      case NO:
        System.out.println("No");
        break;
      case YES:
        System.out.println("Yes");
        break;
      case MAYBE:
        System.out.println("Maybe");
        break;
      case LATER:
        System.out.println("Later");
        break;
      case SOON:
        System.out.println("Soon");
        break;
      case NEVER:
        System.out.println("Never");
        break;
```

```
  public static void main(String args[]) {
    Question q = new Question();

    answer(q.ask());
    answer(q.ask());
    answer(q.ask());
    answer(q.ask());
  }
}
```

Inheritance in Interfaces

- An Interface can extend another interface
  - A class that implements the extended version:
    - Must provide implementation of chain of interface methods
    - Code Example

Default Methods

Development in interfaces starting from JDK 8

Now you can add a default method:

A method with body (Non Abstract Method)

Advantage:

- Without effecting implementation classes, Added Functionality is achieved
- Even Implementation classes can override the concrete method of Interface

Declaring a default method

Only difference between a class method and default method is use of keyword **default**

Default Method

```
public interface MyIF {
  // This is a "normal" interface method declaration.
  // It does NOT define a default implementation.
  int getNumber();

  // This is a default method. Notice that it provides
  // a default implementation.
  default String getString() {
    return "Default String";
  }
}
```

Default Method

```
// Implement MyIF.
class MyIFImp implements MyIF {
  // Only getNumber() defined by MyIF needs to be implemented.
  // getString() can be allowed to default.
  public int getNumber() {
    return 100;
  }
}
```

```
// Implement MyIF.
class MyIFImp implements MyIF {
  // Only getNumber() defined by MyIF needs to be implemented.
  // getString() can be allowed to default.
  public int getNumber() {
    return 100;
  }
}
```

```
// Use the default method.
class DefaultMethodDemo {
  public static void main(String args[]) {

    MyIFImp obj = new MyIFImp();

    // Can call getNumber(), because it is explicitly
    // implemented by MyIFImp:
    System.out.println(obj.getNumber());

    // Can also call getString(), because of default
    // implementation:
    System.out.println(obj.getString());
  }
}
```

Overriding concrete method of Interface to achieve specific function

```
class MyIFImp2 implements MyIF {
  // Here, implementations for both getNumber( ) and getString( ) are provided.
  public int getNumber() {
    return 100;
  }

  public String getString() {
    return "This is a different string.";
  }
}
```

Now, when **getString( )** is called, a different string is returned.

Multiple Inheritance

- Java doesn’t support inheriting from multiple classes
- But to some extent we can achieve it:
  - A class implements more than two interfaces

Multiple Inheritance Issues

- Alpha and Beta Two Interfaces
  - Both have a default method call reset()
- MyClass implements both Alpha and Beta
  - Which reset() will be called?
  - Error will occur
    - To avoid error, MyClass can override the reset() method and it will take preference (Class implementation takes priority)

Multiple Inheritance Issues

- Beta extends Alpha, and both have a method reset()
  - Beta's version will take preference
  - But what if implementation class overrides?

For example, if **Beta** wants to refer to **Alpha**'s default for **reset( )**, it can use this statement:

```
Alpha.super.reset();
```

Use Static Methods in Interface

- We can also add static methods to an interface, like we add them to the classes.
- We can call them directly with Interface name

```java
public interface MyIF {
  // This is a "normal" interface method declaration.
  // It does NOT define a default implementation.
  int getNumber();

  // This is a default method. Notice that it provides
  // a default implementation.
  default String getString() {
    return "Default String";
  }

  // This is a static interface method.
  static int getDefaultNumber() {
    return 0;
  }
}
```