Java interfaces and abstract classes

Interfaces

A Java interface defines a set of methods but does not implement them. A class that implements the interface agrees to implement all of the methods defined in the interface, thereby agreeing to certain behavior. A class can implement multiple interfaces.

You use an interface to define a protocol of behavior that can be implemented by any class anywhere in the class hierarchy. Interfaces are useful for the following:

- Capturing similarities between unrelated classes without artificially forcing a class relationship
- Declaring methods that one or more classes are expected to implement
- Revealing an object's programming interface without revealing its class. (Objects such as these are called *anonymous objects* and can be useful when shipping a package of classes to other developers.)

Abstract classes

An abstract class is a class that may be thought of as between an interface and a regular class. It can contain abstract methods (not implemented) and fully implemented methods. You can never instantiate an abstract class - you need to extend an abstract class to use it.

Notes on Interfaces and Abstract Classes

INTERFACES

- Like a class, an interface can have public or default package access.
- An interface can extend another interface.
- An interface can contain methods, but the following apply:
  - All methods are abstract
  - You can not define implementation of methods in an interface
  - You can include the ``abstract'' qualifier in the method declaration but you do not have to, since it is implied.
  - **native**, **synchronized**, **static** and **final** qualifiers are not allowed.
  - All methods are public.
  - If the access specifier **public** is not specified, it is implied.
- An interface can include fields, but they are all static and final.

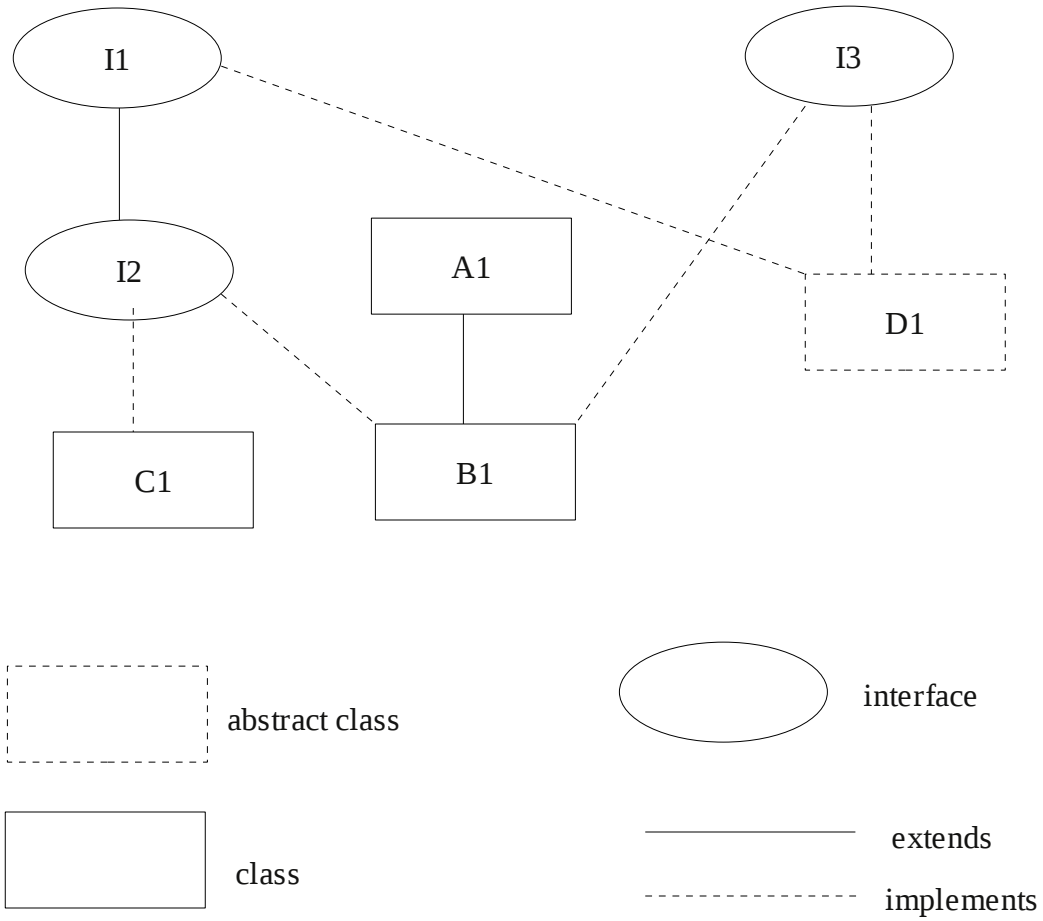
ABSTRACT CLASSES

- Declare a class abstract if it is never to be instantiated.
- The only way to use an abstract class is to extend it.
- Abstract classes can contain abstract as well as fully implemented methods
- When a subclass is instantiated, the abstract super class' constructor is called.
- A class that contains an abstract method is abstract, and MUST be declared abstract. The abstract keyword is repetitious, but required.

Choosing between an interface and abstract superclass

If multiple inheritance is needed, interfaces are the only choice. If that is not an issue, if some of the properties are common to all subclasses and can be implemented in the super class, use an abstract superclass. Otherwise use an interface.

An Example




```
interface I1 {
        void methodI1(); // public static by default
}

interface I2 extends I1 {
        void methodI2(); // public static by default
}

class A1 {
        public String methodA1() {
```

```
                String strA1 = "I am in methodA1 of class A1";
                return strA1;
        }
        public String toString() {
                return "toString() method of class A1";
        }
}

class B1 extends A1 implements I2 {
        public void methodI1() {
                System.out.println("I am in methodI1 of class B1");
        }
        public void methodI2() {
                System.out.println("I am in methodI2 of class B1");
        }
}

class C1 implements I2 {
        public void methodI1() {
                System.out.println("I am in methodI1 of class C1");
        }
        public void methodI2() {
                System.out.println("I am in methodI2 of class C1");
        }
}

// Note that the class is declared as abstract as it does not satisfy the interface contract
abstract class D1 implements I2 {
        public void methodI1() {
        }
        // This class does not implement methodI2() hence declared abstract.
}


public class InterfaceEx {
        public static void main(String[] args) {
                I1 i1 = new B1();
                i1.methodI1(); // OK as methodI1 is present in B1
                // i1.methodI2(); Compilation error as methodI2 not present in I1
                // Casting to convert the type of the reference from type I1 to type I2
                ((I2) i1).methodI2();
                I2 i2 = new B1();
                i2.methodI1(); // OK
                i2.methodI2(); // OK
                // Does not Compile as methodA1() not present in interface reference I1
                // String var = i1.methodA1();
                // Hence I1 requires a cast to invoke methodA1
                String var2 = ((A1) i1).methodA1();
                System.out.println("var2 : " + var2);
```

```
                String var3 = ((B1) i1).methodA1();
                System.out.println("var3 : " + var3);
                String var4 = i1.toString();
                System.out.println("var4 : " + var4);
                String var5 = i2.toString();
                System.out.println("var5 : " + var5);
                I1 i3 = new C1();
                String var6 = i3.toString();
                System.out.println("var6 : " + var6); // It prints the Object toString() method
                Object o1 = new B1();
                // o1.methodI1(); does not compile as Object class does not define
                // methodI1()
                // To solve the probelm we need to downcast o1 reference. We can do it
                // in the following 4 ways
                ((I1) o1).methodI1(); // 1
                ((I2) o1).methodI1(); // 2
                ((B1) o1).methodI1(); // 3
                /*
                 *
                 * B1 does not have any relationship with C1 except they are "siblings".
                 *
                 * Well, you can't cast siblings into one another.
                 *
                 */
                // ((C1)o1).methodI1(); Produces a ClassCastException
        }
}
```

```
linux3:~/CS2/Week5$ java InterfaceEx
I am in methodI1 of class B1
I am in methodI2 of class B1
I am in methodI1 of class B1
I am in methodI2 of class B1
var2 : I am in methodC1 of class A1
var3 : I am in methodC1 of class A1
var4 : toString() method of class A1
var5 : toString() method of class A1
var6 : C1@39172e08
I am in methodI1 of class B1
I am in methodI1 of class B1
I am in methodI1 of class B1
linux3:~/CS2/Week5$
```