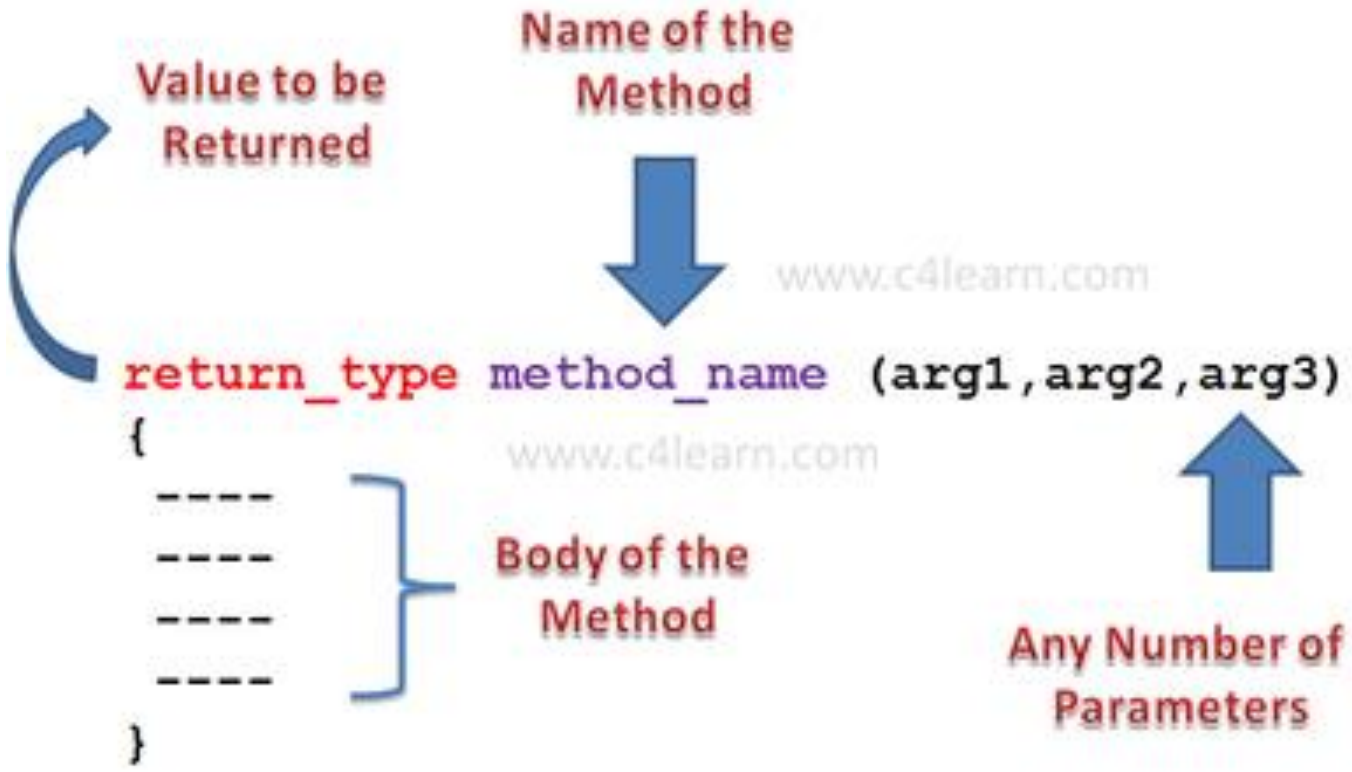
Lecture 5

Abstract classes and interfaces

Java Methods. Let's bring to mind



Abstract – At class level

- A concrete class extending an abstract class should override all the abstract methods.

```
public abstract class Mobile{
        public abstract void call();
}
```

```
public class SmartPhone extends Mobile{
        public void call(){
        }
}
```

The abstract Modifier

- **The abstract class**
  - Cannot be instantiated
  - Should be extended and implemented in subclasses
- **The abstract method**
  - Method signature without implementation

Abstract Classes

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

- Therefore, the following statement, which creates an array whose elements are of GeometricObject type, is correct.

GeometricObject[] geo = new GeometricObject[10];

The Abstract Calendar Class and Its GregorianCalendar subclass

- An instance of java.util.Date represents a specific instant in time with millisecond precision.

- java.util.Calendar is an abstract base class for extracting detailed information such as year, month, date, hour, minute and second from a Date object.

- Subclasses of Calendar can implement specific calendar systems such as Gregorian calendar, Lunar Calendar and Jewish calendar. Currently, java.util.GregorianCalendar for the Gregorian calendar is supported in the Java API.

Interfaces

- An *interface* is a classlike construct that ***contains only constants and abstract methods*** .
- In many ways, an interface is similar to an abstract class, but ***an abstract class can contain variables and concrete methods*** as well as constants and abstract methods.

To distinguish an interface from a class, Java uses the following syntax to declare an interface:

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

Interface is a Special Class

- An interface is treated like a special class in Java. Each interface is compiled into a separate bytecode file, just like a regular class.

- ***Like an abstract class, you cannot create an instance from an interface using the new operator.***

- but in most cases you can use an interface more or less the same way you use an abstract class. For example, ***you can use an interface as a data type for a variable*** , as the result of casting, and so on.

Example of an Interface

use an interface to define a generic compareTo method

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

public interface Comparable {
  public int compareTo(Object o);
}
```

String and Date Classes

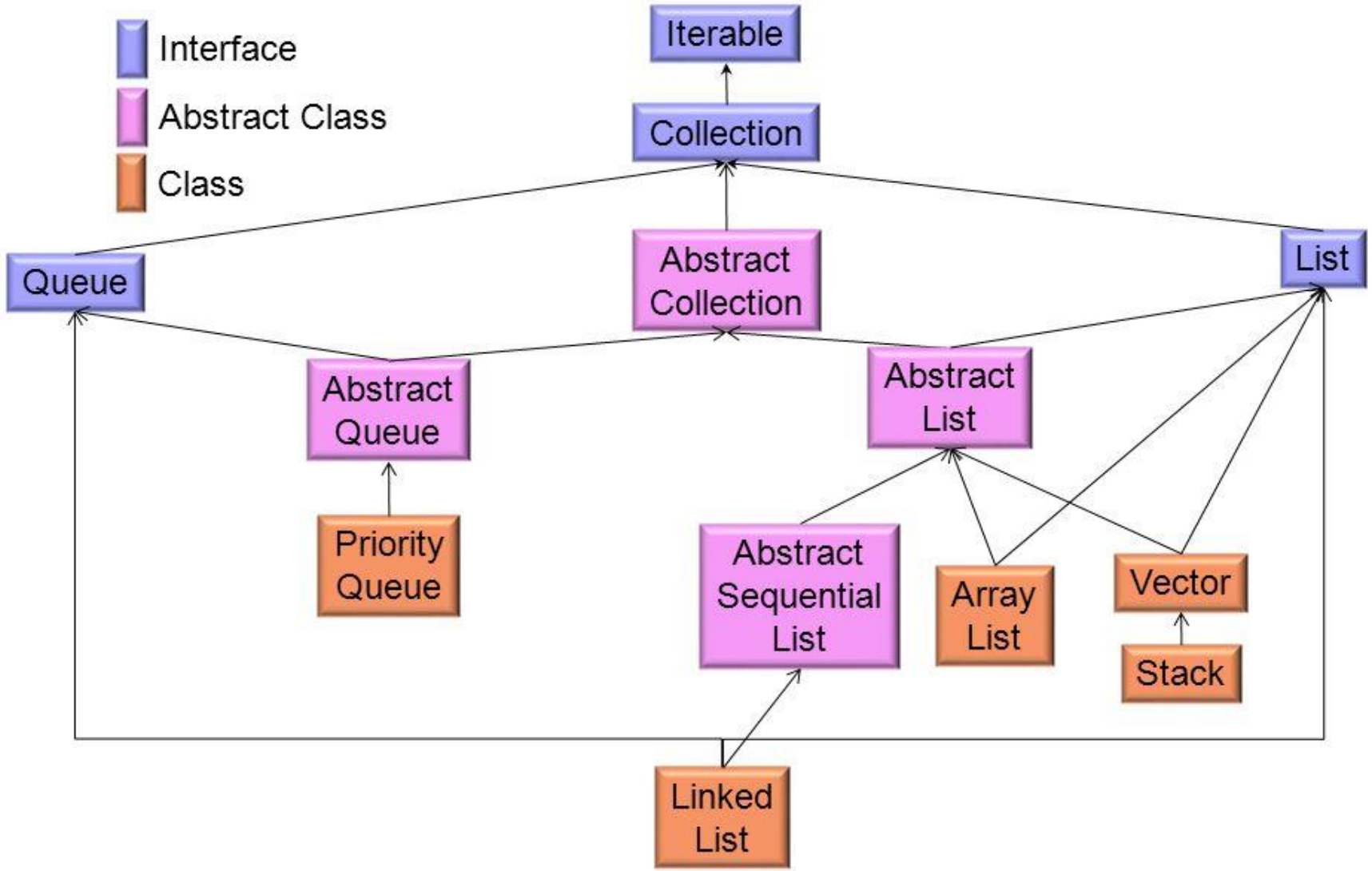
Many classes (e.g., String and Date) in the Java library implement Comparable to define a natural order for the objects. the source code of these classes

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

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

```
new String() instanceof String
new String() instanceof Comparable
new java.util.Date() instanceof java.util.Date
new java.util.Date() instanceof Comparable
```

The Java Collections Framework (Ordered Data Types)



Generic max Method


```java
// Max.java: Find a maximum object
public class Max {
  /** Return the maximum of two objects */
  public static Comparable max
      (Comparable o1, Comparable o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

(a)

```java
// Max.java: Find a maximum object
public class Max {
  /** Return the maximum of two objects */
  public static Object max
      (Object o1, Object o2) {
    if ((Comparable)o1).compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

(b)

```java
String s1 = "abcdef";
String s2 = "abcdee";
String s3 = (String)Max.max(s1, s2);
```

```java
Date d1 = new Date();
Date d2 = new Date();
Date d3 = (Date)Max.max(d1, d2);
```

The return value from the max method is of the Comparable or Object type. So, you need to cast it to String or Date explicitly.

Difference between abstract class and interface

Feature	Interface	Abstract class
Multiple Inheritance	A class may implement several interfaces.	A class may inherit only one abstract class.
Default implementation	An interface is purely abstract, it cannot provide any code, just the signature.	An abstract class can provide complete, default code and/or just the details that have to be overridden.
Access modifiers	An interface cannot have access modifiers for the method, properties etc. Everything is assumed as public.	An abstract class can contain access modifiers for the methods, properties etc.
Core vs. Peripheral	Interfaces are used to define the peripheral abilities of a class. In other words both Human and Vehicle can inherit from a IMovable interface.	An abstract class defines the core identity of a class and there it is used for related objects.
Homogeneity	If various implementations only share method signatures then it is better to use Interfaces.	If various implementations are of the same kind and use common behavior or status then abstract class is better to use.

Abstract class vs Interface (Different)

Abstract class

- To declare an abstract class, use **`abstract`** keyword.

  public abstract class B{

  }
- A class can extend only one abstract class.

  class A extends B{

  }
- In relationship, we say A is B.

Interface

- To declare an interface, use **`abstract`** keyword.

  public interface B{

  }
- A class can implement more than one interface.

  class A implements C, D, E{

  }
- In relationship, we A has C, D, and E.