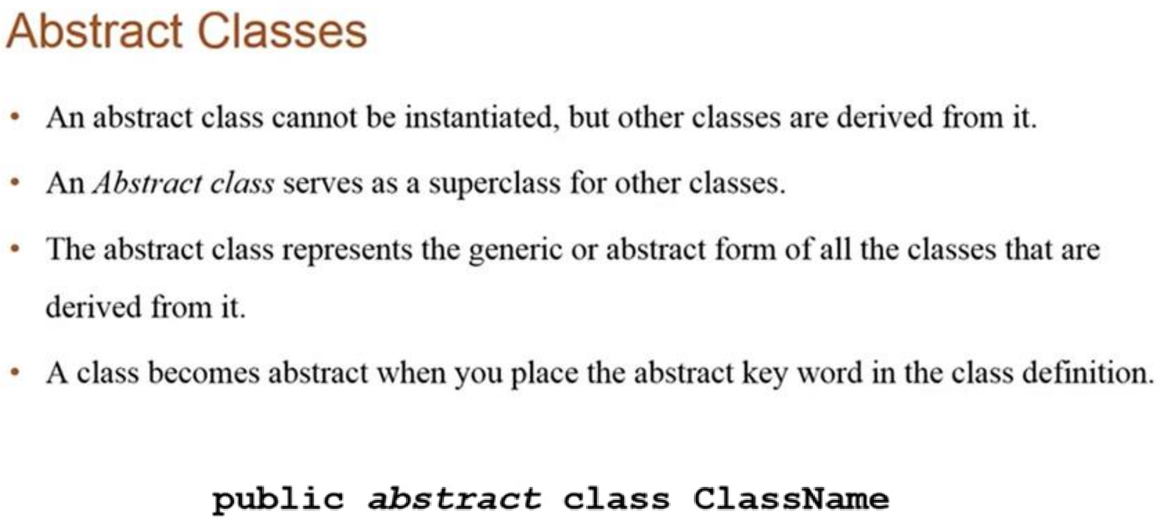
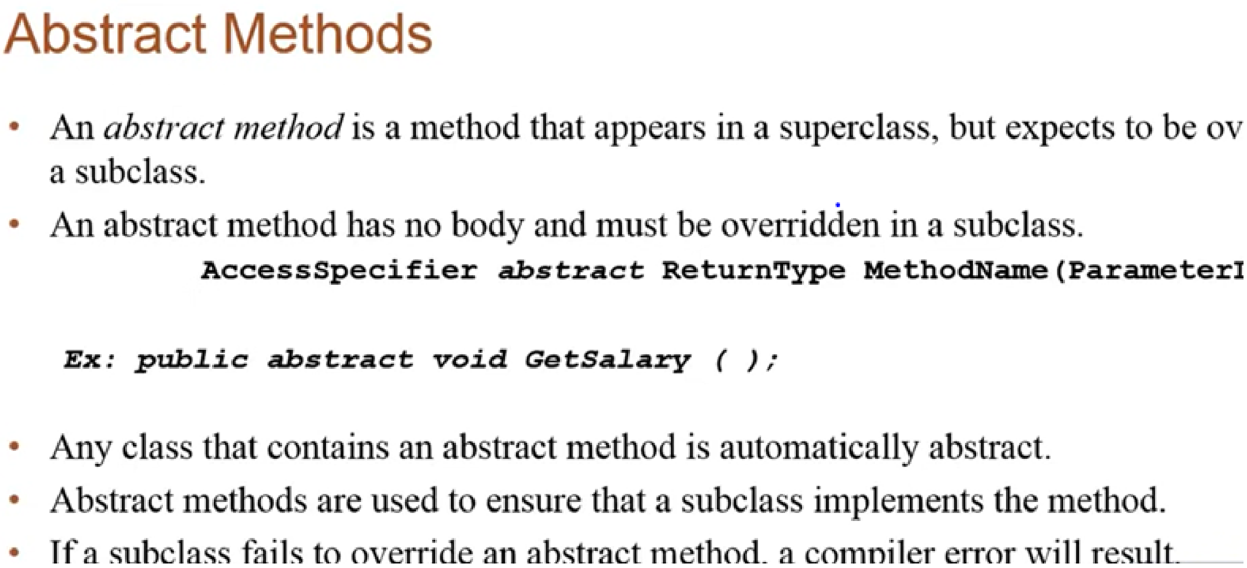


Abstract Classes

- An abstract class cannot be instantiated, but other classes are derived from it.
- An *Abstract class* serves as a superclass for other classes.
- The abstract class represents the generic or abstract form of all the classes that are derived from it.
- A class becomes abstract when you place the abstract key word in the class definition.

```
public abstract class ClassName
```



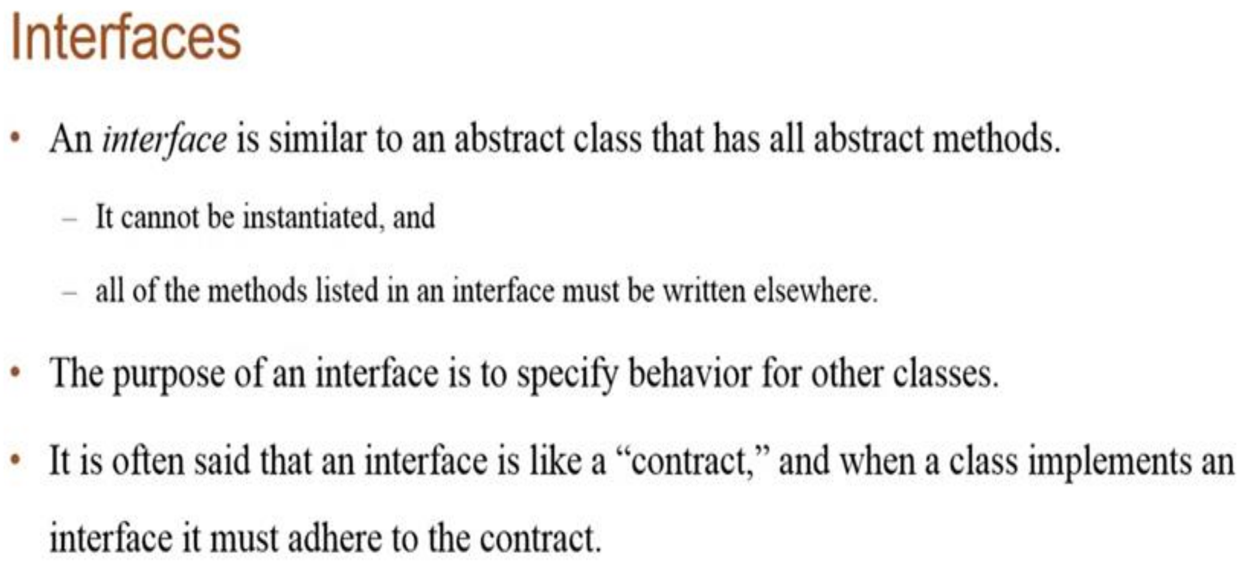
Abstract Methods

- An *abstract method* is a method that appears in a superclass, but expects to be ov a subclass.
- An abstract method has no body and must be overridden in a subclass.

```
AccessSpecifier abstract ReturnType MethodName(ParameterI
```

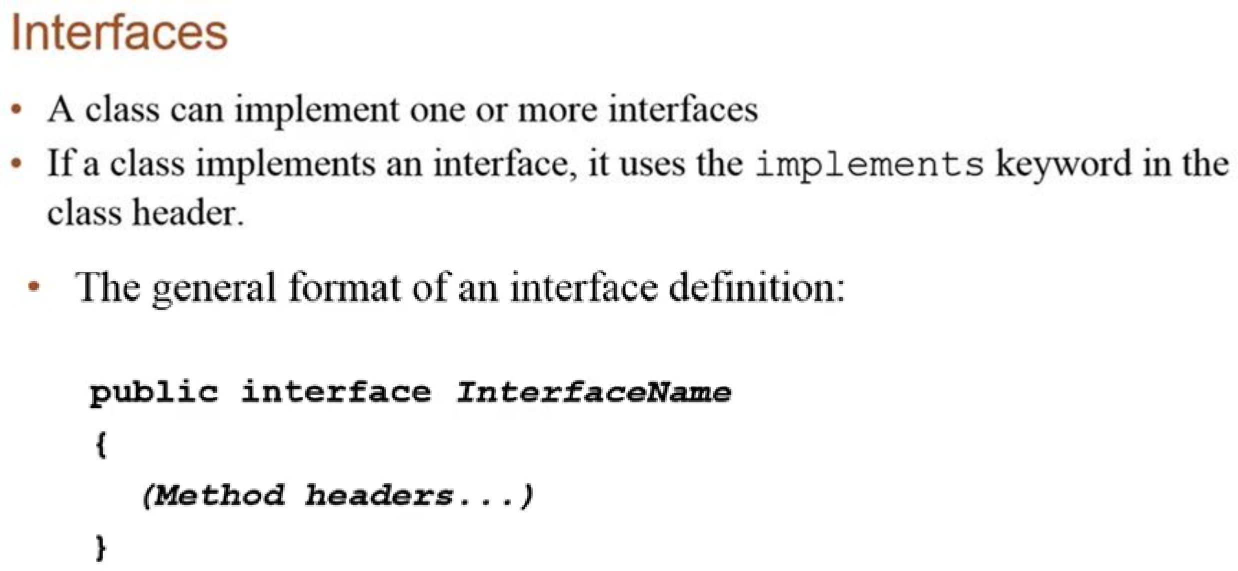
```
Ex: public abstract void GetSalary ( );
```

- Any class that contains an abstract method is automatically abstract.
- Abstract methods are used to ensure that a subclass implements the method.
- If a subclass fails to override an abstract method, a compiler error will result.



Interfaces

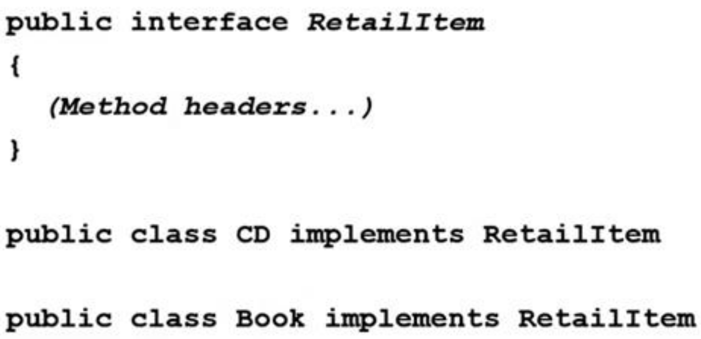
- An *interface* is similar to an abstract class that has all abstract methods.
  - It cannot be instantiated, and
  - all of the methods listed in an interface must be written elsewhere.
- The purpose of an interface is to specify behavior for other classes.
- It is often said that an interface is like a "contract," and when a class implements an interface it must adhere to the contract.



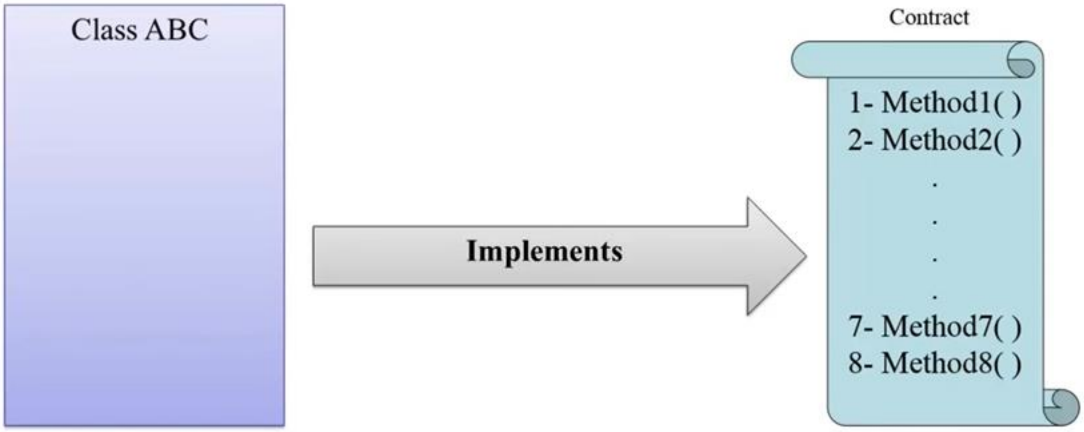
Interfaces

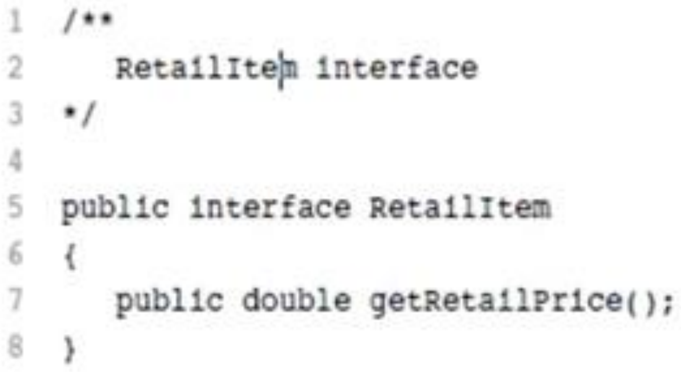
- A class can implement one or more interfaces
- If a class implements an interface, it uses the `implements` keyword in the class header.
- The general format of an interface definition:

```
public interface InterfaceName
{
   (Method headers...)
}
```

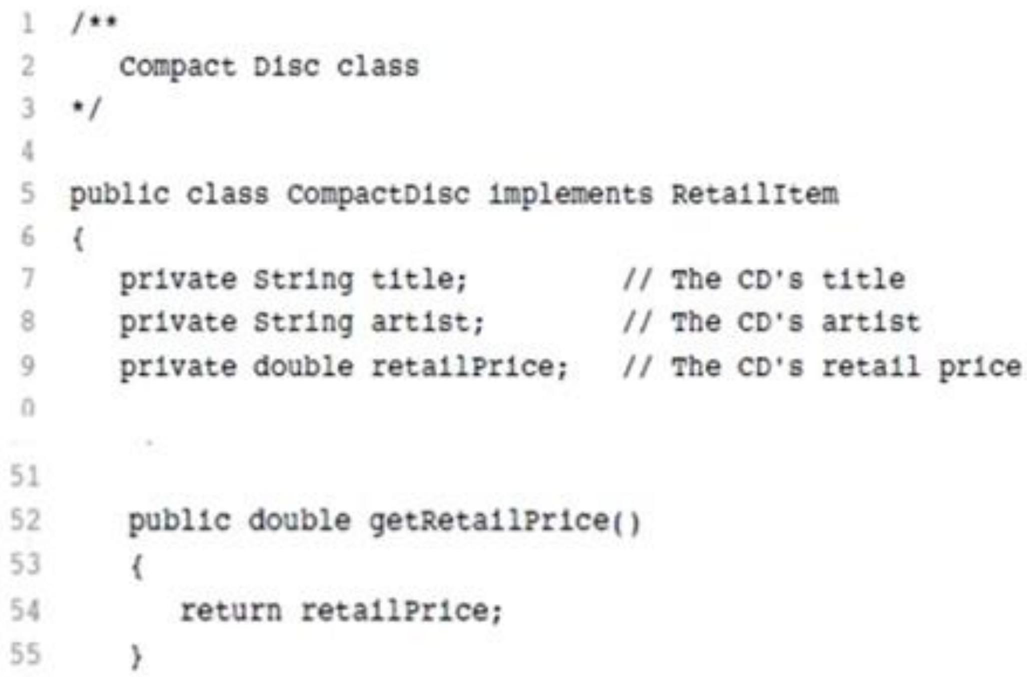


```
public interface RetailItem
{
   (Method headers...)
}

public class CD implements RetailItem

public class Book implements RetailItem
```





```
/**
   RetailItem interface
*/

public interface RetailItem
{
   public double getRetailPrice();
}
```



```
/**
   Compact Disc class
*/

public class CompactDisc implements RetailItem
{
   private String title;         // The CD's title
   private String artist;        // The CD's artist
   private double retailPrice;   // The CD's retail price



    public double getRetailPrice()
    {
       return retailPrice;
    }
```

Enumerated Types

- Known as an `enum`, requires declaration and definition like a class
- **Syntax:**

  `enum` ***typeName*** `{` *one or more enum constants* `}`

- **Definition**:

enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY}

enum CarColor { RED, BLACK, BLUE, SILVER }

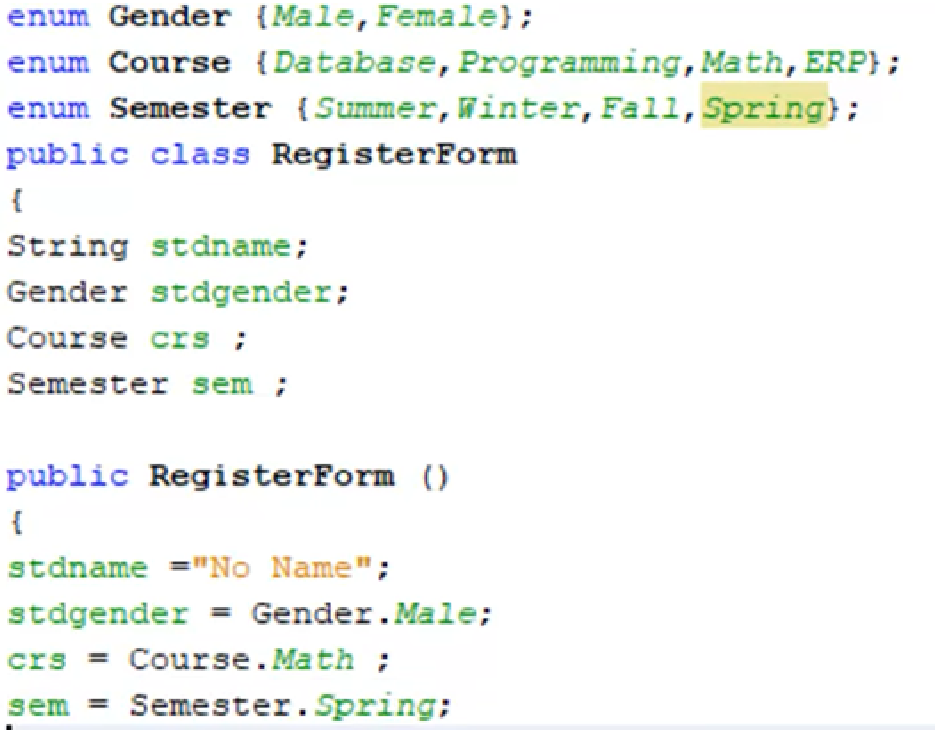
enum CarType { PORSCHE, FERRARI, JAGUAR }

  - **Declaration**:

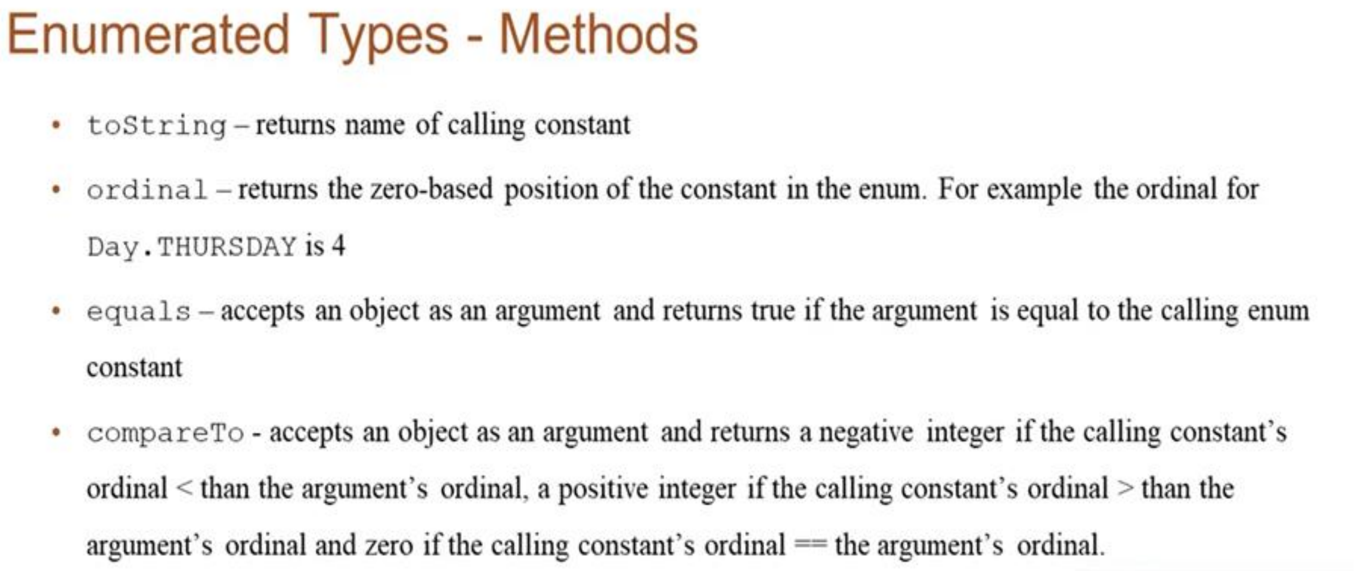
    Day WorkDay; // creates a Day enum

  - **Assignment**:

    Day WorkDay = Day.WEDNESDAY;



```
enum Gender {Male,Female};
enum Course {Database,Programming,Math,ERP};
enum Semester {Summer,Winter,Fall,Spring};
public class RegisterForm
{
String stdname;
Gender stdgender;
Course crs ;
Semester sem ;

public RegisterForm ()
{
stdname ="No Name";
stdgender = Gender.Male;
crs = Course.Math ;
sem = Semester.Spring;
```



Enumerated Types - Methods

- `toString` – returns name of calling constant
- `ordinal` – returns the zero-based position of the constant in the enum. For example the ordinal for `Day.THURSDAY` is 4
- `equals` – accepts an object as an argument and returns true if the argument is equal to the calling enum constant
- `compareTo` - accepts an object as an argument and returns a negative integer if the calling constant's ordinal < than the argument's ordinal, a positive integer if the calling constant's ordinal > than the argument's ordinal and zero if the calling constant's ordinal == the argument's ordinal.