

# Java Programming – Abstract Class & Interface

**Oum Saokosal**

**Master's Degree in information systems, Jeonju  
University, South Korea**

**012 252 752 / 070 252 752**

**oumsaokosal@gmail.com**



# Contact Me

- Tel: 012 252 752 / 070 252 752
- Email: [oumsaokosal@gmail.com](mailto:oumsaokosal@gmail.com)
- FB Page: <https://facebook.com/kosalgeek>
- PPT: <http://www.slideshare.net/oumsaokosal>
- YouTube: <https://www.youtube.com/user/oumsaokosal>
- Twitter: <https://twitter.com/okosal>
- Web: <http://kosalgeek.com>

# Abstract Classes and Interfaces

---

The objectives of this chapter are:

- To explore the concept of abstract classes
- To understand interfaces
- To understand the importance of both.

# What is an Abstract class?

---

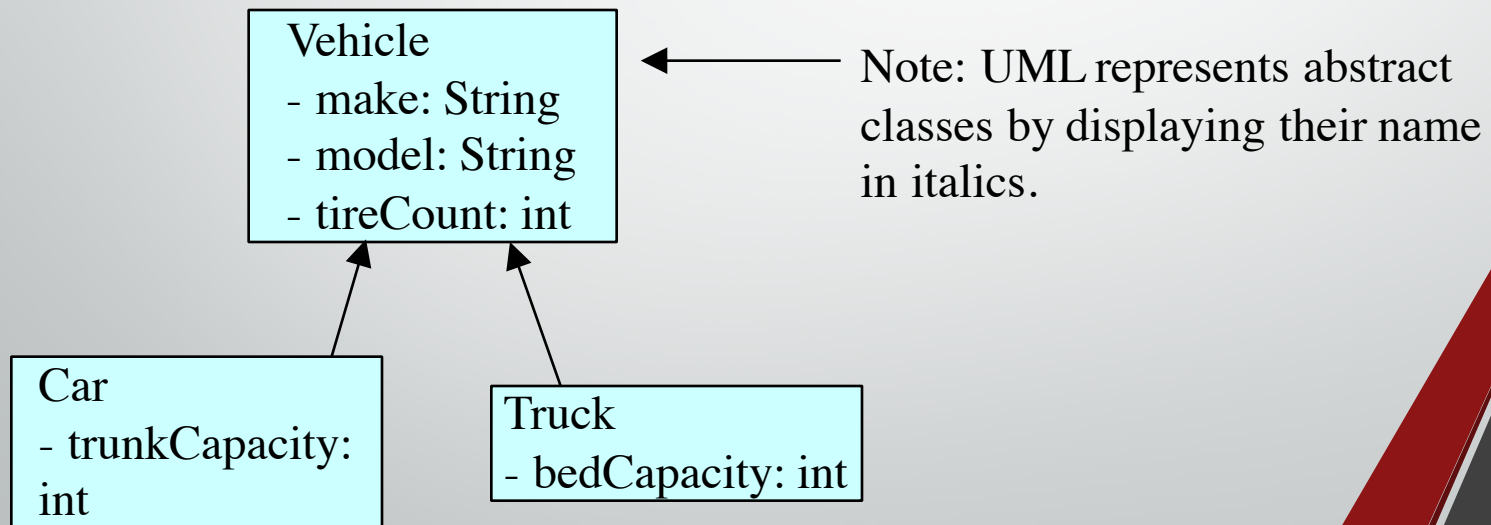
- Superclasses are created through the process called "generalization"
  - Common features (methods or variables) are factored out of object classifications (ie. classes).
  - Those features are formalized in a class. This becomes the superclass
  - The classes from which the common features were taken become subclasses to the newly created super class
- Often, the superclass does not have a "meaning" or does not directly relate to a "thing" in the real world
  - It is an artifact of the generalization process
- Because of this, abstract classes cannot be instantiated
  - They act as place holders for abstraction

# Abstract Class Example

---

- In the following example, the subclasses represent objects taken from the problem domain.
- The superclass represents an abstract concept that does not exist "as is" in the real world.

Abstract superclass:



# What Are Abstract Classes Used For?

---

- Abstract classes are used heavily in Design Patterns
  - Creational Patterns: Abstract class provides interface for creating objects. The subclasses do the actual object creation
  - Structural Patterns: How objects are structured is handled by an abstract class. What the objects do is handled by the subclasses
  - Behavioural Patterns: Behavioural interface is declared in an abstract superclass. Implementation of the interface is provided by subclasses.
- Be careful not to over use abstract classes
  - Every abstract class increases the complexity of your design
  - Every subclass increases the complexity of your design
  - Ensure that you receive acceptable return in terms of functionality given the added complexity.

# Defining Abstract Classes

- Inheritance is declared using the "extends" keyword
  - If inheritance is not defined, the class extends a class called Object

```
public abstract class Vehicle
{
    private String make;
    private String model;
    private int tireCount;
    [...]
}
```

```
public class Car extends Vehicle
{
    private int trunkCapacity;
    [...]
}
```

```
public class Truck extends Vehicle
{
    private int bedCapacity;
    [...]
}
```

Vehicle  
- make: String  
- model: String  
- tireCount: int

Car  
- trunkCapacity: int

Truck  
- bedCapacity: int

Often referred to as  
"concrete" classes

# Abstract Methods

---

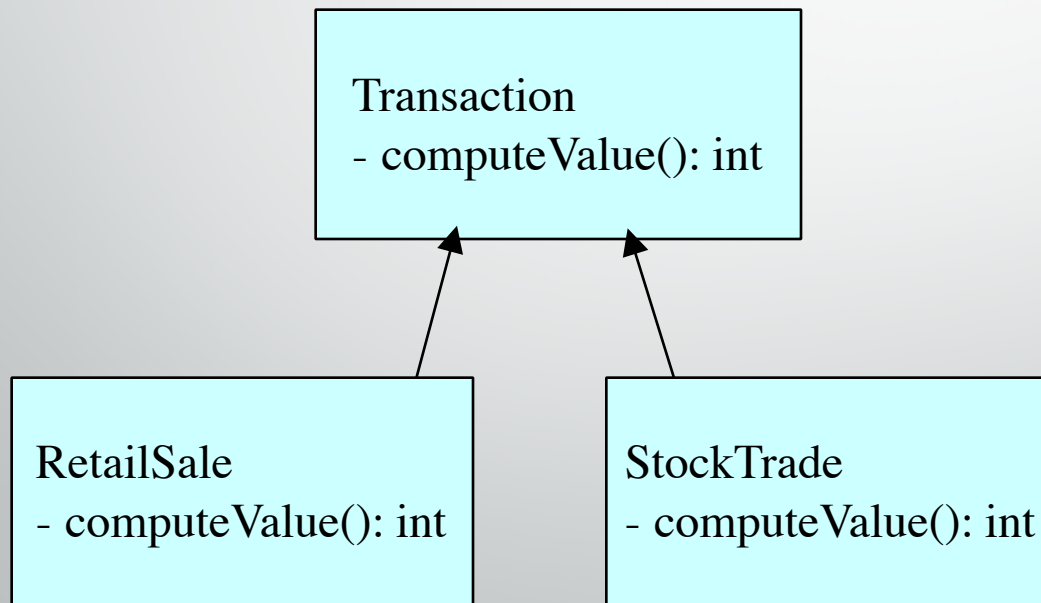
- Methods can also be abstracted
  - An abstract method is one to which a signature has been provided, but no implementation for that method is given.
  - An Abstract method is a placeholder. It means that we declare that a method must exist, but there is no meaningful implementation for that methods within this class
- Any class which contains an abstract method **MUST** also be abstract
  - Any class which has an incomplete method definition cannot be instantiated (ie. it is abstract)
- Abstract classes can contain both concrete and abstract methods.
  - If a method can be implemented within an abstract class, and implementation should be provided.



# Abstract Method Example

---

- In the following example, a Transaction's value can be computed, but there is no meaningful implementation that can be defined within the Transaction class.
  - How a transaction is computed is dependent on the transaction's type
  - Note: This is polymorphism.



# Defining Abstract Methods

- Inheritance is declared using the "extends" keyword
  - If inheritance is not defined, the class extends a class called Object

```
public abstract class Transaction
{
    public abstract int computeValue();
}
```

Note: no implementation

```
Transaction
- computeValue(): int
```

```
public class RetailSale extends Transaction
{
    public int computeValue()
    {
        [...]
    }
}
```

```
RetailSale
- computeValue(): int
```

```
StockTrade
- computeValue(): int
```

```
public class StockTrade extends Transaction
{
    public int computeValue()
    {
        [...]
    }
}
```

# What is an Interface?

---

- An interface is similar to an abstract class with the following exceptions:
  - All methods defined in an interface are abstract. Interfaces can contain no implementation
  - Interfaces cannot contain instance variables. However, they can contain public static final variables (ie. constant class variables)
- Interfaces are declared using the "interface" keyword
  - If an interface is public, it must be contained in a file which has the same name.
- Interfaces are more abstract than abstract classes
- Interfaces are implemented by classes using the "implements" keyword.


# Declaring an Interface

---

In Steerable.java:

```
public interface Steerable
{
    public void turnLeft(int degrees);
    public void turnRight(int degrees);
}
```

When a class "implements" an interface, the compiler ensures that it provides an implementation for all methods defined within the interface.



In Car.java:

```
public class Car extends Vehicle implements Steerable
{
    public int turnLeft(int degrees)
    {
        [...]
    }

    public int turnRight(int degrees)
    {
        [...]
    }
}
```

# Implementing Interfaces

---

- A Class can only inherit from one superclass. However, a class may implement several Interfaces
  - The interfaces that a class implements are separated by commas
- Any class which implements an interface must provide an implementation for all methods defined within the interface.
  - NOTE: if an abstract class implements an interface, it NEED NOT implement all methods defined in the interface. HOWEVER, each concrete subclass MUST implement the methods defined in the interface.
- Interfaces can inherit method signatures from other interfaces.

# Declaring an Interface

---

In Car.java:

```
public class Car extends Vehicle implements Steerable, Driveable
{
    public int turnLeft(int degrees)
    {
        [...]
    }

    public int turnRight(int degrees)
    {
        [...]
    }

    // implement methods defined within the Driveable interface
```

# Inheriting Interfaces

---

- If a superclass implements an interface, its subclasses also implement the interface

```
public abstract class Vehicle implements Steerable
{
    private String make;
    [...]
```

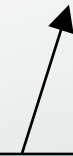
```
public class Car extends Vehicle
{
    private int trunkCapacity;
    [...]
```

```
public class Truck extends Vehicle
{
    private int bedCapacity;
    [...]
```

Vehicle  
- make: String  
- model: String  
- tireCount: int

Car  
- trunkCapacity: int

Truck  
- bedCapacity: int



# Multiple Inheritance?

---

- Some people (and textbooks) have said that allowing classes to implement multiple interfaces is the same thing as multiple inheritance
- This is NOT true. When you implement an interface:
  - The implementing class does not inherit instance variables
  - The implementing class does not inherit methods (none are defined)
  - The Implementing class does not inherit associations
- Implementation of interfaces is not inheritance. An interface defines a list of methods which must be implemented.



# Interfaces as Types

---

- When a class is defined, the compiler views the class as a new type.
- The same thing is true of interfaces. The compiler regards an interface as a type.
  - It can be used to declare variables or method parameters

```
int i;  
Car myFleet[];  
Steerable anotherFleet[];  
  
[...]  
  
myFleet[i].start();  
  
anotherFleet[i].turnLeft(100);  
anotherFleet[i+1].turnRight(45);
```

# Abstract Classes Versus Interfaces

---

- When should one use an Abstract class instead of an interface?
  - If the subclass-superclass relationship is genuinely an "is a" relationship.
  - If the abstract class can provide an implementation at the appropriate level of abstraction
- When should one use an interface in place of an Abstract Class?
  - When the methods defined represent a small portion of a class
  - When the subclass needs to inherit from another class
  - When you cannot reasonably implement any of the methods