JAVA @17

Interfaces

Objective

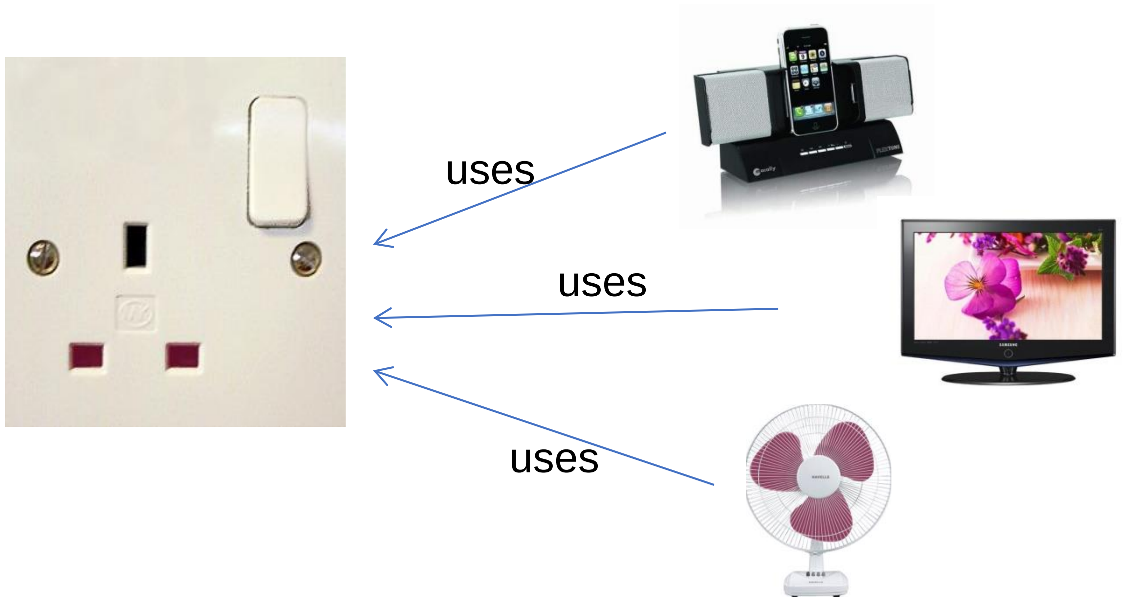
After completing this session you will be able to understand,

- Introduction to Interface
- Implement an Interface
- Why do we use Interfaces
- Interface vs Abstract Classes vs Class
- Interface and Polymorphism
- Java8 Default method in Interface
- Java8 Static method in Interface
- Marker Interface

Interface

What is Interface?

Plug Point is a interface which helps different devices to connect to the electrical line to draw power



So interface is a common protocol that all devices need to adhere for them to operate.

In software terminology the interface is an contract between the class and the outside world, and this contract is enforced at the build time by the compiler

Interface

What is Interface?

When **all methods** in a class are ***abstract***, the class can be declared as an Interface.

An Interface defines a contract for a classes to implement the behavior. Similar to the plug point defines the contract for all devices to draw power from electricity lines.

Interface in real world

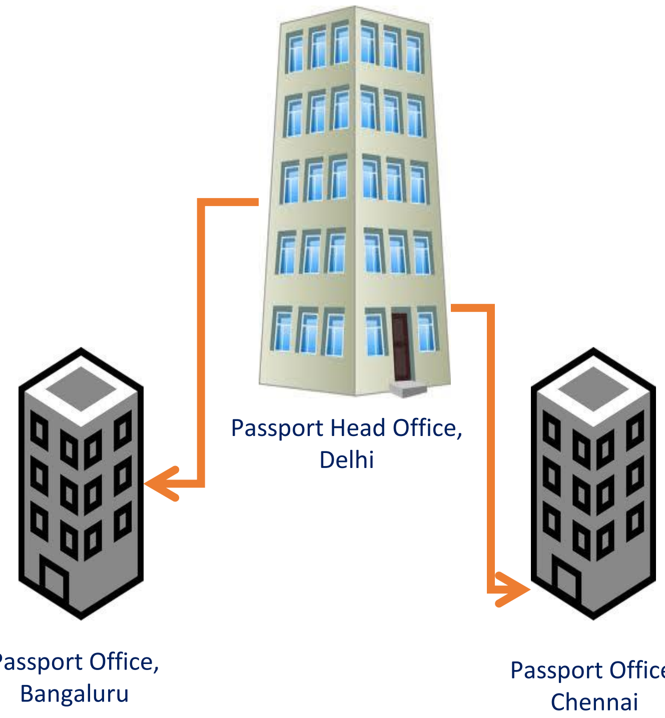
Scenario: Taking the passport example if in the issuance both the ***verification of photo identity*** and ***final approval*** process needs to be performed by the local offices. The head office at Delhi defines only the protocol/ process for photo identity verification process and approval but the actual process will be performed by the local offices.

Delhi Office,

- Defines a protocol/process that all local passport office should follow for verifying the photo identity & final approval.

Bangaluru,

- Adheres to the protocol and performs photo identification using ration card.
- Approves the document as slated by the HO.



Chennai,

- Adheres to the protocol and performs photo identification using driver license.
- Approves the document as slated by the HO.

Analogy between Interface and Passport office

Passport Head Office	→	Interface
Local passport Offices	→	Sub Class
Photo Identity Verification	→	**Protocol:** interface methods

NOTE: No people can directly reach Delhi head office for passport application rather they should always go to the local offices in appropriate cities which will follow the process defined by head office to issue passport.

Similarly Interface cannot be instantiated only the classes implementing it can be instantiated and the implemented methods invoked.

Develop an Interface

How to create an Interface?

To create an interface, the ‘class’ keyword should be replaced with the ‘interface’ keyword

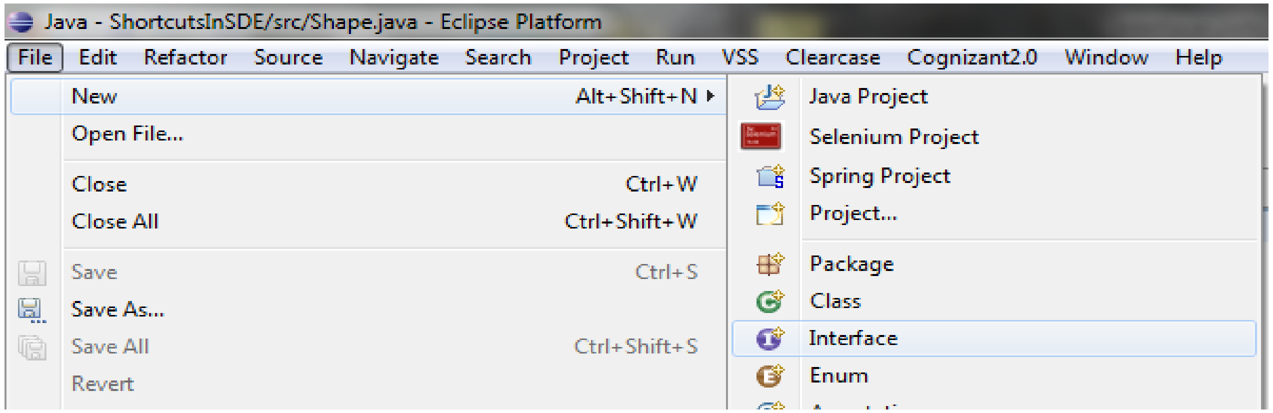
- To create a Shape interface, instead of

  public class Shape

  public interface Shape

- To create an interface using IDE, select

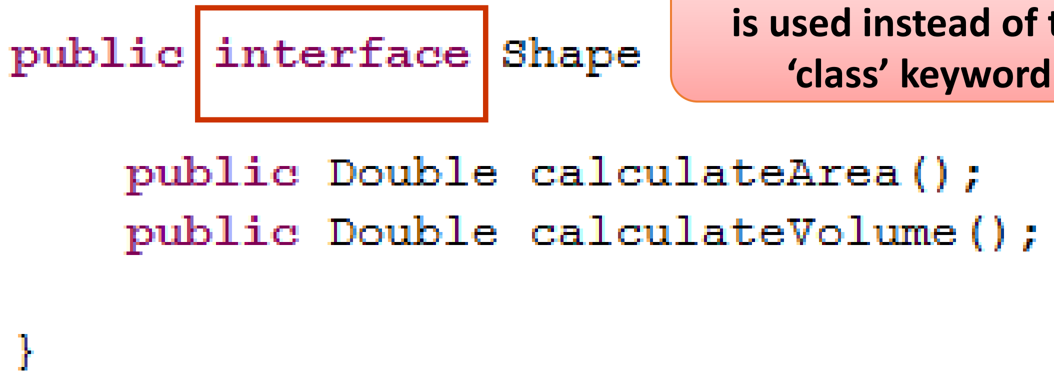
  File → New → Interface



How to implement an Interface?

How to implement an Interface?

Step 1: Create an Interface



```
public interface Shape

     public Double calculateArea();
     public Double calculateVolume();

}
```

The 'interface' keyword is used instead of the 'class' keyword

Abstract Methods

How to implement an Interface?

How to implement an Interface?

Step 2: Create the implementation class using the '**implements**' keyword

```
public class Circle implements Shape {

    Double radius = 5.0;

    @Override
    public Double calculateArea() {
        Double area = 3.14 * radius * radius;
        return area;
    }

    @Override
    public Double calculateVolume() {
        Double volume = (4/3) * 3.14 * radius * radius * radius;
        return volume;
    }
}
```

'**implements**' keyword is used to indicate that **class Circle** is the **implementation** class for the **interface 'Shape'**

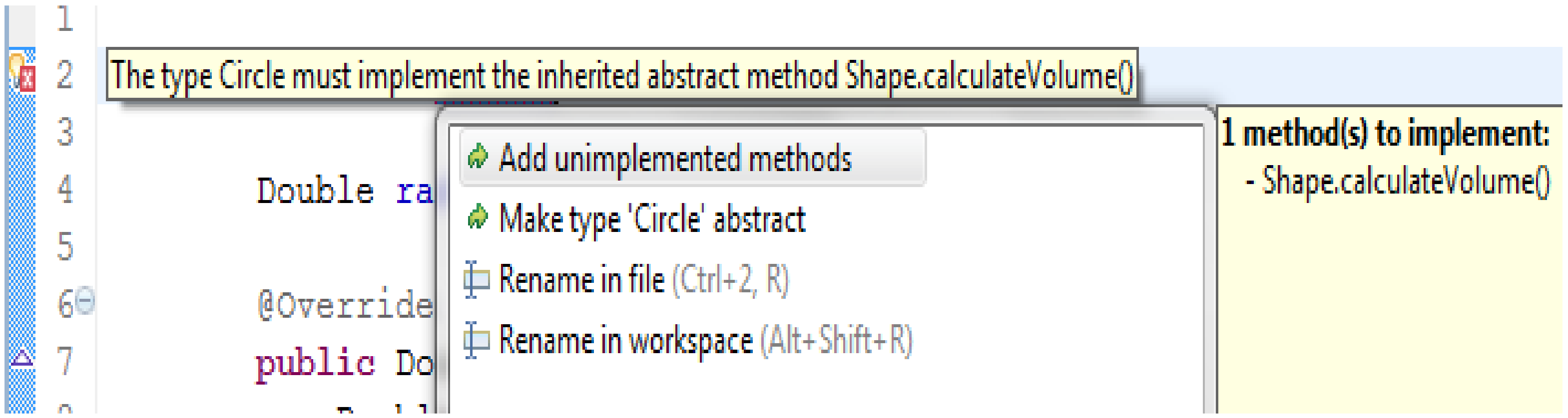
The actual implementation logic for the methods in the interface is provided in this class

Similarly, other shape objects can be created by interface programming by implementing generic **Shape interface.**

Implementing Interface

Some facts about Implementing Interfaces

- A Class can implement any number of interfaces.
- When your class tries to implement an interface, always make sure that you implement all the methods of that interface, or else, you would get the following error at compile time
- Implementing class can have its own methods (which are not present in the Interface class).
- Implementing class can also extend a single super class or abstract class.



The type Circle must implement the inherited abstract method Shape.calculateVolume()

```
Double ra
@Override
public Do
```

Add unimplemented methods
Make type 'Circle' abstract
Rename in file (Ctrl+2, R)
Rename in workspace (Alt+Shift+R)

1 method(s) to implement:
- Shape.calculateVolume()

Why do we use Interface?

Reason #1:

To reveal the programming interface of an object without revealing its implementation:

- This is the concept of encapsulation, hiding implementation from consumers.

Analogy to remote example: Only the plug point is exposed. The method of drawing current using the internal wiring of the electrical appliance is not known to the user

- The implementation can change without affecting the caller of the interface

Analogy to remote example: Any changes required to the wiring in the plug point can be done without any changes in the TV/Fan/Ipod Dock

- The caller does not need the implementation at compile time

Analogy to remote example: The manufacture of the plug point is not related to the manufacture of TV/Fan/Ipod Dock

Why do we use Interface?

Reason #2:

To have unrelated classes implement similar methods (behavior)

The blueprint of the classes will be defined in the interface and all the classes implementing that interface must follow the same blue print (implement the same methods from the interface)

Analogy to the remote example:

All the electrical appliances should adhere to the norms specified by the plug point. Ex. Voltage range from – 220V to 240V

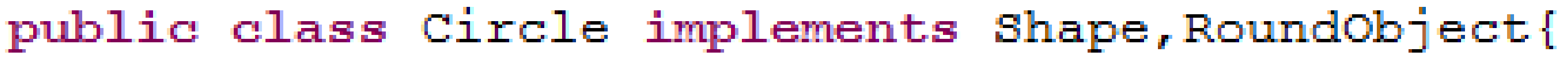
Why do we use Interface?

Reason #3:

To model multiple inheritance:

A class can implement multiple interfaces while it can extend only one class.

Example:



```
public class Circle implements Shape,RoundObject{
```

The concrete class ***Circle*** can implement two(or more) interfaces ***Shape*** and ***RoundObject***.

Here, the class ***Circle*** should implement all the methods declared in both the interfaces

The Java programming language does not support multiple inheritance, but interfaces provide an alternative

Example - Interface

Let us all create interfaces and implementation classes:

1. Create an interface IVehicle with the below methods:

- drive();
- turnLeft();
- brake();

2. Create another interface IPublicTransport with a method

•getNumberOfPeople();

3. Create a class MotorisedVehicle.java with a method checkMotor() which prints the message "The motor of the vehicle is in good condition"

4. Create a class Car.java which extends the MotorisedVehicle class and implements the IVehicle interface. This method should print appropriate messages in the implemented methods. (Ex. "The car is in brake mode" etc)

5. Create a class Train.java which implements both the IVehicle and IPublicTransport interfaces. The implemented methods should print appropriate messages (Ex. "The train is turning left" etc)

Interface vs Abstract classes

Interface	Abstract Classes
All methods of an interface are implicitly abstract, they can only have method declaration.	Abstract classes can have both abstract (method declaration) and concrete methods (methods with implementation).
Can only define constants which are by default *public final static.* Example: int COUNTRY_CODE=1;	Fields with values can be declared. Fields with values declared in abstract classes can be changed in sub class
A class can implement several interfaces.	A class may extend only one abstract class.
Interface definition begins with keyword “interface”, so it is of type interface.	Abstract classes begins with keyword “abstract class”, so it is of type class.
Interface have no implementation and needs to be ‘*implemented*’.	Abstract classes needs to be *extended*.

Interface vs Abstract classes

When to use Interfaces?

Interfaces can be used when common functionalities have to be implemented differently across multiple classes. The user can mandate the use of these common functionalities to all the classes that implement the interface.

When to use abstract classes?

Abstract classes can be used when

- some implemented functionalities are common between classes (this reduces duplicate code)
- some functionalities need to be implemented in sub classes that extend the abstract class

Point to remember: A concrete class can extend only one super class whether that super class is either concrete or abstract class.
But, a concrete class can implement multiple interfaces and at the same time extend one super class

Interface as a Type

How to instantiate an interface?

You cannot create an instance from an Interface.

You can only define an Interface type and assign it to the instance of the concrete class which implemented the interface

Interface as a type

Interface as a Type:

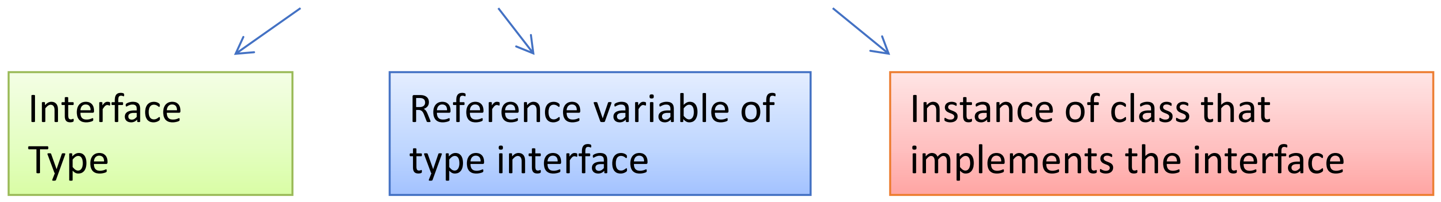
When you define a new Interface, you are defining a new reference type.

If you define a reference variable whose type is an Interface, then any object you assign to it must be an instance of the class that implements the interface

Example:

Person class implements the IPerson interface. You can do:

```
IPerson piVar = new Person();
```


Interface Type
Reference variable of type interface
Instance of class that implements the interface

Example – Declaring Interface reference

Let us now learn to invoke methods of an interface implementation using the same code that we have developed for IVehicle.

Let us learn how to do the below things:

1. Invoke the drive and brake methods of the car object
2. Invoke the checkMotor method on the car object
3. Invoke the drive and brake methods of the train object
4. Invoke the getNumberOfPeople on the train object

Interface – Lend a Hand

To Invoke the brake methods of the car object,

Step 1: Create a InterfaceDemo.java class with a main method to perform all the required actions

Step 2: Create a reference variable of the IVehicle Interface

IVehicle vehicleObj1

Step 3: Assign the reference variable to an instance of the Car Object

IVehicle vehicleObj1 = new Car();

Step 4: Invoke the **drive** and **brake** methods using the interface reference variable

vehicleObj1.drive();

It is your turn to do the rest of the coding !!

Hint: The checkMotor method can be called only on the Car object and
The getNumber of people can be called on the Train object with reference variable pointing to the IPublicTransport interface

Solution - Interface

Solution

```
public class InterfaceDemo {

    public static void main (String args[]){

        // Invoking the drive and brake method on the car object
        IVehicle vehicleObj1 = new Car();
        vehicleObj1.drive();
        vehicleObj1.brake();

        //Invoking the checkmotor method on the car object
        Car car = new Car();
        car.checkMotor();

        //Invoking the drive and brake method on the train object
        IVehicle vehicleObj2 = new Train();
        vehicleObj2.drive();
        vehicleObj2.brake();

        //Invoking the getNumberOfPeople on the train object
        IPublicTransport publicTransObj = new Train();
        publicTransObj.getNumberOfPeople();
    }
}
```

Interface vs Class

Differences between Interface and a Class:

- The methods of an interface are all abstract methods, they cannot have bodies.
- You cannot create an instance from an interface.

For **Example**:

> ***IPerson piVar = new IPerson();*** will throw an ERROR

- An interface can only be implemented by other classes or extended by other interfaces

Developer faces a problem again

Mr. Max, a Java developer from Comcast has a problem !!

He has developed an **interface** called **IVehicle** which can drive and brake.

There are **50 classes** which **implement** this interface.

His client has now asked him to include **a 3rd method** called **accelerate** of his concrete **classes**

If Max makes this change in IVehicle interface, then all 50 classes will break because they do not implement all methods of interface anymore

How can Max enforce this new method in his 25 classes alone without disturbing the other classes?

Max Solution

Use Inheritance of Interfaces

Solution: Create a new Interface **IAdvancedVehicle** that extends the **IVehicle** interface and add the new method in the new interface.

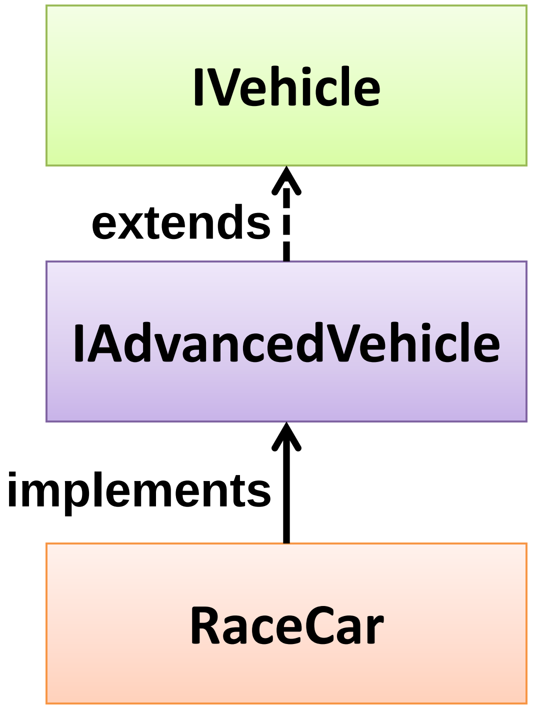
Now, users of your code can choose to continue to apply the old interface or upgrade to the new interface.

People who use the old interface **IVehicle,** need not change their code as the interface is untouched.

Inheritance among Interfaces

Inheritance among Interfaces:

Even though interfaces are not part of the class hierarchy, they can have inheritance relationship among themselves.



```
public interface IVehicle {

     public void drive();
}


public interface IAdvancedVehicle extends IVehicle {

     public void accelerate();

}
public class RaceCar implements IAdvancedVehicle {

     @Override
     public void accelerate() {
          System.out.println("Accelarate");

     }
     @Override
     public void drive() {
          System.out.println("drive");
     }
}
```

Interface and Polymorphism

Interface and Polymorphism:

Interfaces exhibit polymorphism, because interfaces can be used for declaring reference and based on the type of object instance created and injected into the reference the appropriate objects will be triggered.

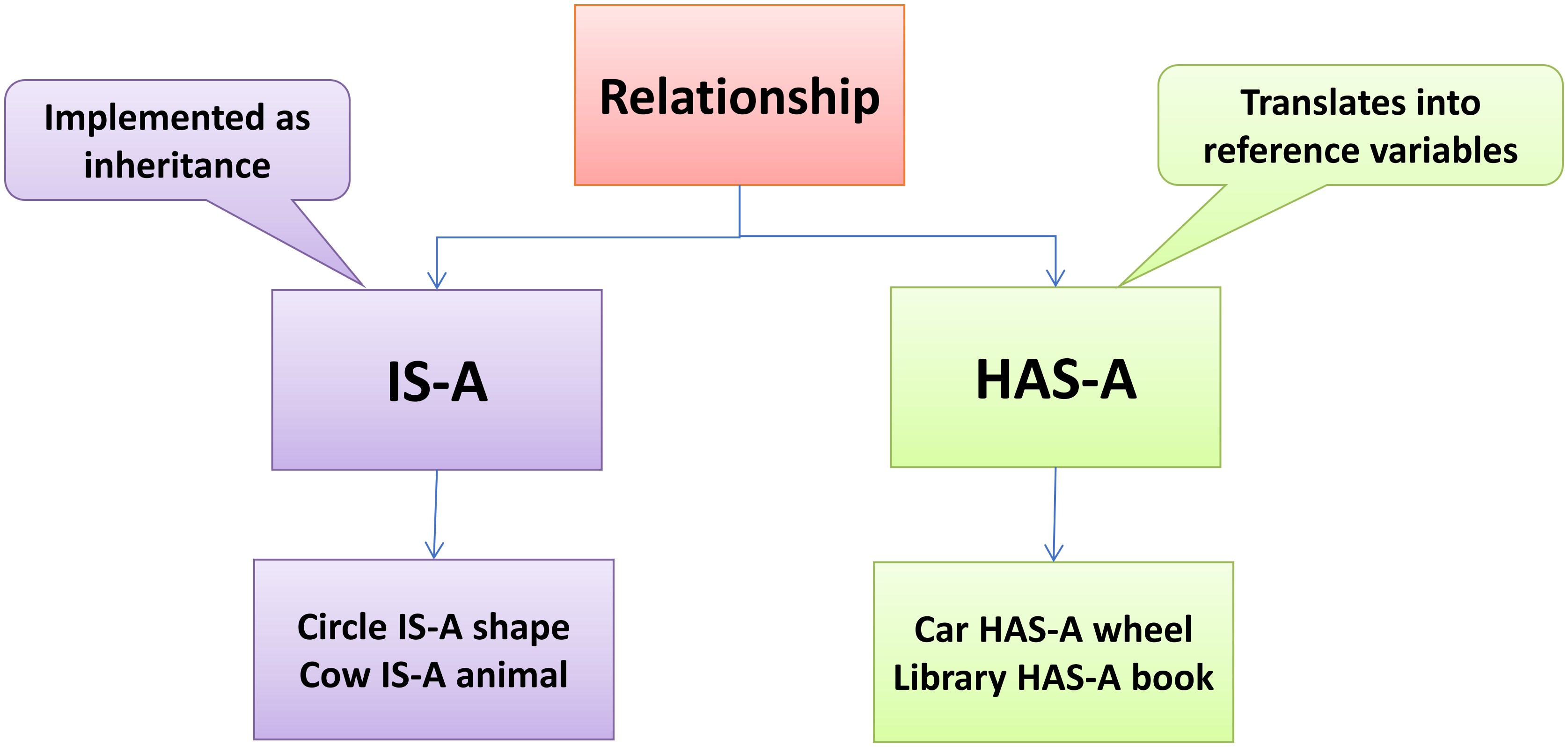
Example:

```
ICar car1= new Honda();
ICar car2= new BMW();
car1.drive();
car2.drive();
```

Based on the objects (**Honda,BMW**) created and injected into the interface reference (**car1, car2**) the appropriate objects method will be invoked.

IS-A and HAS-A Relationship

Java represents two types of relationship



Using IS-A relationship

IS-A relationship:

- When one class **inherits** from another, you say that the subclass **extends** the super class.
- When you want to know if one thing should extend another, use the **IS-A** test.
- **Examples** for IS-A relationship:

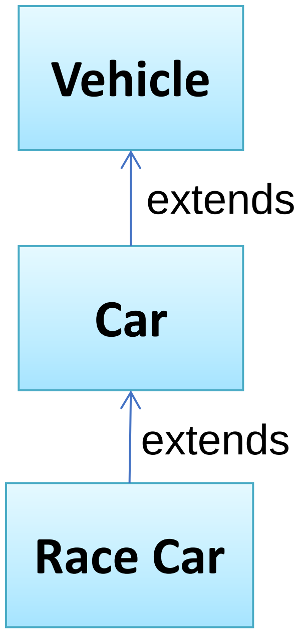
  Triangle IS-A Shape

  Green IS-A Color
- **Do not** apply inheritance if the sub class and super class do not pass the IS-A test

Using IS-A relationship

IS-A relationship:

- The IS-A test works anywhere in the inheritance tree



Car IS-A Vehicle

RaceCar IS-A Car

RaceCar IS-A Vehicle

Note: The IS-A relationship works in only one direction

Example:

Car IS-A vehicle makes sense, but Vehicle IS-A car does not make sense

Using HAS-A relationship

HAS-A relationship:

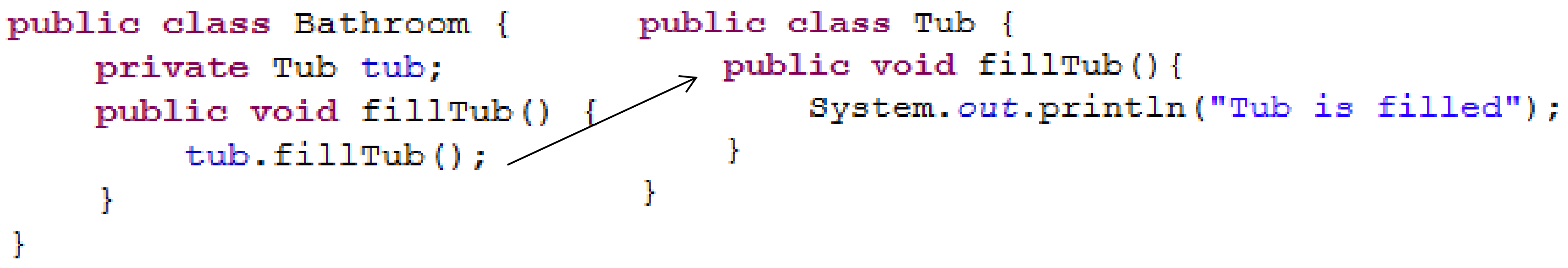
When two classes are related by not through inheritance, then you say that the two classes are joined by HAS-A relationship

Example:

Bathroom HAS-A tub

Tub HAS-A bubble

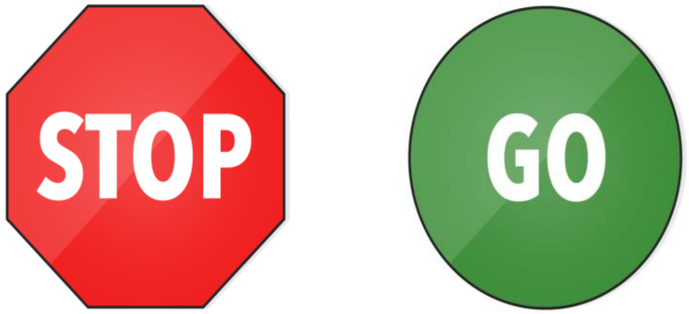
The HAS-A relationship can be translated into reference variables.



```java
public class Bathroom {
    private Tub tub;
    public void fillTub() {
        tub.fillTub();
    }
}
```

```java
public class Tub {
    public void fillTub(){
        System.out.println("Tub is filled");
    }
}
```

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is an Interface?
- How to implement the interface?
- Why do we use interfaces?
- Difference between interface and abstract classes?
- How to test if inheritance is required?

Thank you

You have successfully completed

Interface