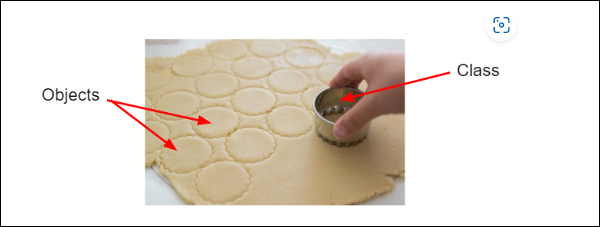
# Class

## what is class in java

In Java, a class is a blueprint or template for creating objects that share common properties and behavior. It defines the attributes (data) and methods (functions) that an object of that class can have and perform.



Classes are the fundamental building blocks of object-oriented programming in Java. Each object that is created from a class is known as an instance of that class.

For example, suppose you want to create a program that models a car. You could define a class called "Car" that includes attributes such as make, model, and year, as well as methods such as start, accelerate, and brake. Then, you can create multiple objects of the Car class, each with its own values for the make, model, year, etc.

Here is an example of a simple Car class in Java:

public class Car {

// instance variables

String make;

String model;

int year;

// instance methods

public void start() {

System.out.println("The " + make + " " + model + " is starting.");

}

public void accelerate() {

System.out.println("The " + make + " " + model + " is accelerating.");

}

public void brake() {

System.out.println("The " + make + " " + model + " is braking.");

}

}

This class defines three attributes - make, model, and year - and three methods - start, accelerate, and brake. Once you have defined the class, you can create multiple Car objects, each with its own values for the attributes. For example:

Car myCar = new Car();

myCar.make = "Toyota";

myCar.model = "Corolla";

myCar.year = 2020;

myCar.start(); // Output: The Toyota Corolla is starting.

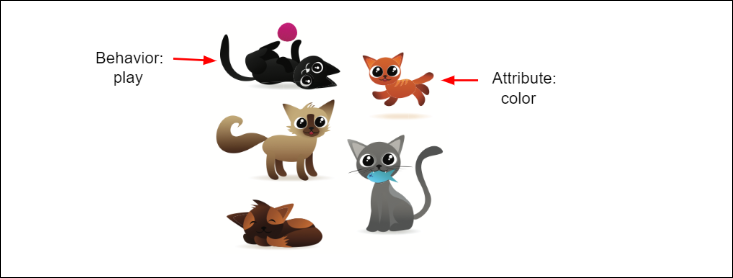
myCar.accelerate(); // Output: The Toyota Corolla is accelerating.

myCar.brake(); // Output: The Toyota Corolla is braking.

This is just a simple example, but classes can become much more complex and include many more attributes and methods.

You can think of a class as the type or classification. Each type can have **attributes** (the object’s properties or what it knows about itself) and **behaviors** (what an object does). In Java code, the attributes are written as **instance variables** in the class, and the behaviors are written as **methods**.

The following picture has lots of cats (objects of the type cat). They are all different, but they share the same attributes and behaviors that make up a cat. They are all **instances** of a cat with different values for their attributes. Name some of the attributes and behaviors of the cats below. For example, the color (attribute) of the first cat is black (attribute value) and it is playing (behavior).



**Object**: At runtime, when the Java Virtual Machine (JVM) encounters the

new keyword, it will use the appropriate class to make an object which is an

instance of that class. That object will have its own state, and access to all of

the behaviors defined by its class.

Car carObject = new Car();

Is like

int value =10;

Where int type, value is variable name and 10 is value assigned.

As as, Car is a type of carObject, where carObject is variable and the value is Car() object.

We create object using new keyword.

**State (instance variables)** Each object (instance of a class) will have its

own unique set of instance variables as defined in the class. Collectively, the

values assigned to an object's instance variables make up the object's state.

// instance variables

String make;

String model;

int year;

**Behavior (methods)** When a programmer creates a class, she creates methods for that class. Methods are where the class' logic is stored. Methods are

where the real work gets done. They are where algorithms get executed, and

data gets manipulated.

start()

brake()

accelerate()

## How to compile and run java class using package

To compile and run a Java program that uses packages, you need to follow these steps:

Create a directory structure that matches your package hierarchy. For example, if your package name is "com.example", create a directory called "com" and inside that, create a directory called "example".

Save your Java source file(s) in the appropriate directory, using the fully qualified package name as the directory path. For example, if your package is "com.example" and your Java file is called "MyClass.java", save it as "com/example/MyClass.java".

Open a command prompt or terminal window and navigate to the directory where your Java file is located.

Compile your Java source file using the javac command and the -d option to specify the output directory for the compiled class files. For example:

javac Hello.java

javac -d . MyClass.java

javac -d output a/Hello.java

This will create the class file in the current directory, but you can specify a different directory if you prefer.

Run your Java program using the java command and the fully qualified name of the class that contains the main method. For example:

java Hello

java com.example.MyClass

java -cp output a.Hello

This will execute the main method in the MyClass class. Note that you must use the fully qualified class name, including the package name, when you run the program.

## Java Naming Convension

Java naming conventions are guidelines for naming Java classes, interfaces, methods, variables, and constants. These conventions help to make the code more readable, maintainable, and understandable. Here are some common Java naming conventions:

**Class names**: Start with a capital letter and use CamelCase. For example, MyClass or PersonDetails.

**Interface names**: Start with a capital letter and use CamelCase, just like class names. For example, MyInterface or ListIterator.

**Method names**: Start with a lowercase letter and use camelCase. For example, calculateTotal() or getData().

**Variable name**s: Start with a lowercase letter and use camelCase. For example, firstName or totalAmount.

**Constant names:** Use all capital letters and separate words with underscores. For example, MAX\_VALUE or PI.

**Package names**: Use lowercase letters and separate words with periods. For example, com.mycompany.mypackage.

**Boolean variable** names: Start with "is" or "has" and use camelCase. For example, isFinished or hasChildren.

**Parameter names**: Start with a lowercase letter and use camelCase. For example, userName or startDate.

**Enum type names**: Start with a capital letter and use CamelCase. For example, DayOfWeek or Color.

Following these naming conventions can help to make your code more readable and maintainable. They also help to prevent naming conflicts and make it easier for other developers to understand and work with your code.

## Legal Identifiers

In Java, an identifier is a name given to a variable, method, class, interface, or other programming element. Here are the rules for legal identifiers in Java:

Identifiers must begin with a letter (A-Z or a-z), the underscore (\_) character, or the dollar sign ($) character.

After the first character, identifiers can include any combination of letters, numbers, underscores, and dollar signs.

Identifiers must not be a keyword or reserved word in Java. For example, you cannot use the identifier public because it is a keyword.

Identifiers are case-sensitive. For example, myVariable and myvariable are two different identifiers.

Identifiers should be descriptive and meaningful, and follow the naming conventions recommended by Java.

Here are some examples of legal identifiers in Java:

int myVariable;

String firstName;

MyClass myClass;

calculateTotal();

MAX\_VALUE;

\_isValid;

$price;

And here are some examples of illegal identifiers in Java:

3DModel // cannot begin with a number

public // cannot use a keyword

my-variable // cannot use hyphens

It's important to follow the rules for legal identifiers in Java, as using illegal identifiers will result in a syntax error and prevent your program from compiling.

**Examples of legal and illegal identifiers follow, first some legal identifiers:**

int \_a;

int $c;

int \_\_\_\_\_\_2\_w;

int \_$;

package com.icici.model; //first statement, 0 or 1 //path

// com/icici/model/Customer.java - developer has to create

//command : javac -d target com/icici/model/Customer.java

// target/com/icici/model/Customer.class

// -d is destination

//run

//java -cp target com.icici.model.Customer

// -cp class path

import java.util.Date; //scanner

class Customer{

int accountNumber; //instance variable

String firstName; //instance variable

String lastName; //instance variable

static final String BANK\_NAME ="Bank of Baroda"; //constant

public int getAccountNumber(){ //instance method / method

return accountNumber;

}

public static void main(String[] hello){

Date date = new Date();

//java.util.Date

//java.sql.Date

System.out.println("Customer ");

}

}

//com.icici.model

///class Account{}

//com.icici.domain

//class Account{}

**The following are illegal:**

int :b;

int -d;

int e#;

int .f;

int 7g;

**Package**:

it is an encapsulation (grouping ) machanism to group related things into a single component

1. Modularity of the application
2. Naming conflict , unique identification
3. We can achieve security for our component, Class declared default can be accessed within the package.

Package pack1; // valid but not recommended , use unique name such as domain name

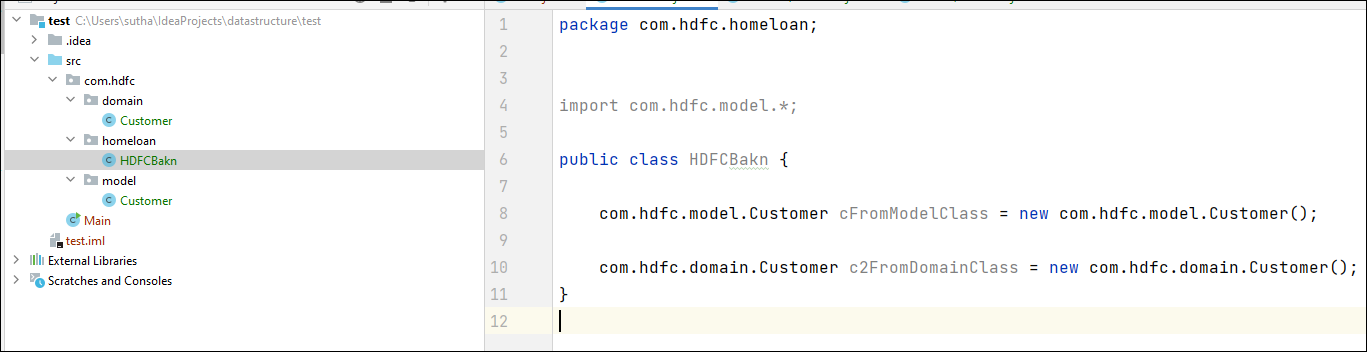
//universally accepted convention is to use domain name.

com.icicbank.loan.housing.Account; // valid and highly recommended.

package com.icici.model; //first statement in the file, there can be 0 or max 1 package statement is allowed in one java file.

|  |  |
| --- | --- |
| package com.icici.model;  class Customer{  int accountNumber; //instance variable  String firstName; //instance variable  String lastName; //instance variable  static final String BANK\_NAME ="Bank of Baroda"; //constant        public int getAccountNumber(){ //instance method / method  return accountNumber;  }    public static void main(String[] hello){  System.out.println("Customer ");  } | Create folder com/icici/model  Keep Customer.java in model folder  Compile command :  javac -d target com/icici/model/Customer.java  where -d is destination, to keep bytecode files.  We should not mix source and binaries in same location.  Once you compile: below folder and class file will be created: target/com/icici/model/Customer.class  To run:  java -cp target com.icici.model.Customer  where -cp is classpath to instruct where bytecode file. |

|  |  |
| --- | --- |
| package com.icici.service;  import com.icici.model.\*;  public final class CustomerService{  public static void main(String[] hello){  Customer customer = new Customer();  }  } | Create service folder  keep CustomerService file.  And create Customer object. |
| class Outer{    protected class Inner{    }    } | When you compile Outer$Inner.class file will be generated. |
| class A{    }  class B{    }  class C{    }  class D{  public static void main(String[] args) {  System.out.println("D");  }  }  class E{  public static void main(String[] args) {  System.out.println("E");  }  }  class Test101 {  public static void main(String[] args) {  System.out.println("Test101");  }  } | Compile and see how may class file will be generated.  Filename could be anything, as long as class is not public    If I save file with name, Ravi.java  Then A.class B.class ..Test101 .class files will be generated.  When we are executing correspond class will be executed.    Javac Ravi.java  Java Test101 --> this file will be executed main method. |
| class A{}  public class B{} | If class is public, then name of the class file must be match with the  Public class name.     * A single file could not have two public class within it., It will be a   Compile time error.     * A file contains multiple class declaration but file name must   Matched with one public class name.     * If all classes are nonpublic then file name could be anything.     Compiler error: class C public, it should be declared in C.java |
| public class Test102 {    public static void main(String[] arsgs){  ArrayList<String> list=new ArrayList<String>();  }  } | Test102.java:6: error: cannot find symbol  ArrayList<String> list=new ArrayList<String>();  ^  symbol: class ArrayList  location: class Test102  Test102.java:6: error: cannot find symbol  ArrayList<String> list=new ArrayList<String>();  ^  symbol: class ArrayList  location: class Test102  2 errors        How the compiler know that class ArrayList is missing,  Because it is declared as class, so this kind of error is showing. |
| public class Test102 {    public static void main(String[] arsgs){  ArrayList();  }  } | Test102.java:6: error: cannot find symbol  ArrayList();  ^  symbol: method ArrayList()  location: class Test102  1 error      Now see the error carefully,  ArrayList is declared as method, so this error. |
| public class Test102 {    public static void main(String[] arsgs){  java.util.ArrayList<String> list=new java.util.ArrayList<String>();  }  } | Now we have provided fully qualified name,  Now compile able to identify which class to load. |
| import java.util.ArrayList;  public class Test102 {    public static void main(String[] arsgs){  ArrayList<String> list=new ArrayList<String>();  ArrayList<String> list=new ArrayList<String>();  ArrayList<String> list=new ArrayList<String>();  ArrayList<String> list=new ArrayList<String>();  ArrayList<String> list=new ArrayList<String>();  ArrayList<String> list=new ArrayList<String>();  }  } | Now it is happily able to compile.  Readability is improved. Rather than calling with fully qualified name.       1. Explicit class import    1. Import java.util.ArrayList 2. Implicit class import    1. Import java.util.\*        1. Import java.util.ArrayList; 2. Import java.util.\*; 3. Import java.util.ArrayList.\*; 4. Import java.util;       Performance vise below both are same. Just for readability vise improved. To See which class we are using that can be identified  Import java.util.ArrayList;  Import java.util.\*;      Recommendation vise Import java.util.ArrayList; is good. |
|  |  |



If Customer file is present in two package then we need to provide fully qualified name as below to avoid conflicts.

|  |
| --- |
| public class HDFCBakn {   com.hdfc.model.Customer cFromModelClass = new com.hdfc.model.Customer();   com.hdfc.domain.Customer c2FromDomainClass = new com.hdfc.domain.Customer(); } |

There can be only one public class per source code file.

■ Comments can appear at the beginning or end of any line in the source code

file; they are independent of any of the positioning rules discussed here.

■ If there is a public class in a file, the name of the file must match the name

of the public class. For example, a class declared as public class Dog { }

must be in a source code file named Dog.java.

■ If the class is part of a package, the package statement must be the first line

in the source code file, before any import statements that may be present.

■ If there are import statements, they must go between the package statement

(if there is one) and the class declaration. If there isn't a package statement,

then the import statement(s) must be the first line(s) in the source code file.

If there are no package or import statements, the class declaration must be

the first line in the source code file.

■ import and package statements apply to all classes within a source code file.

In other words, there's no way to declare multiple classes in a file and have

them in different packages, or use different imports.

■ A file can have more than one nonpublic class.

Source File Declaration Rules (Exam Objective 1.1) 11

■ Files with no public classes can have a name that does not match any of the

classes in the file

**modifiers for class:**

**Access modifiers**: public, protected, private.

**Non-access modifiers** (including strictfp, final, and abstract).

|  |  |
| --- | --- |
| class MyClass{} | MyClass is default class. Accessible within the same package, outside of the package not accessible. |
| public class MyClass{} | class is public, accessible outside of the package. Accessible to all package. |
| private class A{} | class cannot be private |
| protected class B{} | class cannot be protected |
| Class OuterClass{  private class Inner1{}  protected class Inner2{}  } | Inner class can be private and protected. |
| --------  file name : A.java  -------  package a;  class A{  int getMethod1(){return 1;}  }  -----------  file name : B.java  -----------  package b;  import a;  class B{  int getMethod1(){return 2;}  int process(){  A aObject = new A();  aObject.getMethod1(); //not accessble    C cObject = new C();  cObject.getMethod1();  }  } | Create A.java in a package.  Create B class in b package.  Try to create object of A inside B class.  Where class is default  Then try adding class A with public and rerun |
| --------  file name : A.java  -------  package a;  class A{  public int getMethod1(){return 1;}  }  -----------  file name : B.java  -----------  package b;  import a;  class B{  int getMethod1(){return 2;}  int process(){  A aObject = new A();  aObject.getMethod1(); //not accessble    C cObject = new C();  cObject.getMethod1();  }  } | If class is not visible then public method will not be visible. Like public method in class A. |
| --------  file name : A.java  -------  package a;  public class A{  public int getMethod1(){return 1;}  }  --------  file name : C.java  -------  package c;  public class C{  public int getMethod1(){return 1;}  }  -----------  file name : B.java  -----------  package b;  import a;  import c;  class B{  int getMethod1(){return 2;}  int process(){  A aObject = new A();  aObject.getMethod1();    C cObject = new C();  cObject.getMethod1();  }  } |  |

|  |  |
| --- | --- |
| Inheritence  class Parent{  public void useCar(){}  }  class Child extends Parent{  } | Using inheritance , parent public method will be accessible to child, private method of parent will not be accessible to child. |
| -------------  inheritence  -------  abstract class Vehical{  public abstract void run();    public void horn(){  "pom-pom";  }  }  class Car extends Parent{    public void run(){  //how to run  }    //overriding  public void horn(){  "pum-pum";  }  } | Class can be abstract, where object of abstract class not be created,  Child class has to provide implementation of run method. |
| public final class CustomerService{  public static void main(String[] hello){  Customer customer = new Customer();  }  } | final class where child of parent class cannot be created. |
| abstract class Car {  private double price;  private String model;  private String year;  public abstract void goFast();  public abstract void goUpHill();  public abstract void impressNeighbors();  // Additional, important, and serious code goes here  } | Create object of Car object  Car c = new Car();  See the compilation error. |