Introduction to Programming

Java – Basics

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

January, 2021

What is Java?

Java was designed for the development of software for consumer electronics.

What is Java?

Java was designed for the development of software for consumer electronics.

Then what it turns out?

What is Java?

Java was designed for the development of software for consumer electronics.

Then what it turns out?

Java is a general-purpose, platform-independent, object-oriented programming language

History

"I'm just an observer of Java" – Dennis Ritchie.

History

"I'm just an observer of Java" – Dennis Ritchie.

1991: Sun Microsystems initiates project led by James Gosling

The father of java

History

"I'm just an observer of Java" – Dennis Ritchie.

1991: Sun Microsystems initiates project led by James Gosling

The father of java

1991: Initial name given *Oak* by the *Green Team*

History

"I'm just an observer of Java" – Dennis Ritchie.

1991: Sun Microsystems initiates project led by James Gosling

The father of java

1991: Initial name given *Oak* by the *Green Team*

1992: The *Green Team* announced applications

History

"I'm just an observer of Java" – Dennis Ritchie.

1991: Sun Microsystems initiates project led by James Gosling

The father of java

1991: Initial name given *Oak* by the *Green Team*

1992: The *Green Team* announced applications

1995: Renamed to java

History

"I'm just an observer of Java" – Dennis Ritchie.

1991: Sun Microsystems initiates project led by James Gosling

The father of java

1991: Initial name given *Oak* by the *Green Team*

1992: The *Green Team* announced applications

1995: Renamed to java

1995: *Netscape Navigator* Internet browser incorporates java technology

History

"I'm just an observer of Java" – Dennis Ritchie.

1991: Sun Microsystems initiates project led by James Gosling

The father of java

1991: Initial name given *Oak* by the *Green Team*

1992: The *Green Team* announced applications

1995: Renamed to java

1995: *Netscape Navigator* Internet browser incorporates java technology

1997: Sun distinguishes between SDK and JRE

History

"I'm just an observer of Java" – Dennis Ritchie.

1991: Sun Microsystems initiates project led by James Gosling

The father of java

1991: Initial name given *Oak* by the *Green Team*

1992: The *Green Team* announced applications

1995: Renamed to java

1995: *Netscape Navigator* Internet browser incorporates java technology

1997: Sun distinguishes between SDK and JRE

Influenced by: Ada 83, C++, C#, Eiffel, Generic Java, Mesa, Modula-3, Oberon, Objective-C, UCSD Pascal, Smalltalk, etc.

Why java is called java?

Java means

Why java is called java?

Java means coffee!!!

Why java is called java?

Java means coffee!!!

The limitations of java

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– Bjarne Stroustrup.

The limitations of java

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– Bjarne Stroustrup.

- Bytecodes slow down the execution

The limitations of java

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– Bjarne Stroustrup.

- Bytecodes slow down the execution
- No control of the programmer on memory management and garbage collection

The limitations of java

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– Bjarne Stroustrup.

- Bytecodes slow down the execution
- No control of the programmer on memory management and garbage collection
- Floating point arithmetic is not fully supported

The limitations of java

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– Bjarne Stroustrup.

- Bytecodes slow down the execution
- No control of the programmer on memory management and garbage collection
- Floating point arithmetic is not fully supported
- Does not support operator overloading

The limitations of java

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– Bjarne Stroustrup.

- Bytecodes slow down the execution
- No control of the programmer on memory management and garbage collection
- Floating point arithmetic is not fully supported
- Does not support operator overloading
- No 'Templates' yet

The limitations of java

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– Bjarne Stroustrup.

- Bytecodes slow down the execution
- No control of the programmer on memory management and garbage collection
- Floating point arithmetic is not fully supported
- Does not support operator overloading
- No 'Templates' yet
- No 'Function pointers' or 'blocks'

The limitations of java

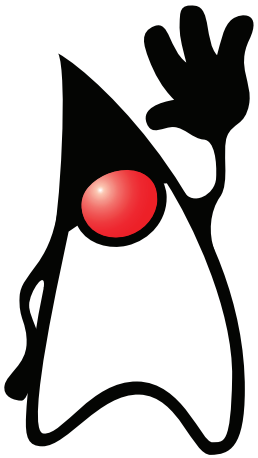
"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– Bjarne Stroustrup.

- Bytecodes slow down the execution
- No control of the programmer on memory management and garbage collection
- Floating point arithmetic is not fully supported
- Does not support operator overloading
- No 'Templates' yet
- No 'Function pointers' or 'blocks'
- Basic classes in class library are lacking important features

Features of java

- General-purpose
- Platform-independent
- Object-oriented
- Robust and secure
- Distributed
- Multithreaded
- Dynamic
- Extensible
- etc.



General-purpose

Simple and diverse applications – works on PCs, Laptops, mobiles, etc.

But NOT iPad/iPhone!!!

General-purpose

Simple and diverse applications – works on PCs, Laptops, mobiles, etc.

But NOT iPad/iPhone!!!

"Java's not worth building in. Nobody uses Java anymore. It's this big heavyweight ball and chain."

– Steve Jobs.

Platform-independent

Source code $\xrightarrow{\textit{compiler}}$ **bytecode** $\xrightarrow{\textit{interpreter}}$ **machine code**

Platform-independent

Source code $\xrightarrow{\textit{compiler}}$ **bytecode** $\xrightarrow{\textit{interpreter}}$ **machine code**

The size of the primitive data types are machine-independent

Object-oriented

"The great thing about Object Oriented code is that it can make small, simple problems look like large, complex ones"

– Anonymous.

Object-oriented

"The great thing about Object Oriented code is that it can make small, simple problems look like large, complex ones"

– Anonymous.

Objects are the basic runtime entities in an object-oriented system ... objects encapsulate data and method

Other features

- **Robust and secure:** Java has strict compile time and run time checking for data types and it is strongly typed

Other features

- **Robust and secure:** Java has strict compile time and run time checking for data types and it is strongly typed
- **Distributed:** It has the ability to share both data and programs and access remote objects

Other features

- **Robust and secure:** Java has strict compile time and run time checking for data types and it is strongly typed
- **Distributed:** It has the ability to share both data and programs and access remote objects
- **Multithreaded:** It can handle multiple tasks simultaneously through multiprocess synchronization

Other features

- **Robust and secure:** Java has strict compile time and run time checking for data types and it is strongly typed
- **Distributed:** It has the ability to share both data and programs and access remote objects
- **Multithreaded:** It can handle multiple tasks simultaneously through multiprocess synchronization
- **Dynamic:** It can dynamically link new class libraries

Other features

- **Robust and secure:** Java has strict compile time and run time checking for data types and it is strongly typed
- **Distributed:** It has the ability to share both data and programs and access remote objects
- **Multithreaded:** It can handle multiple tasks simultaneously through multiprocess synchronization
- **Dynamic:** It can dynamically link new class libraries
- **Extensible:** It supports functions (known as *native methods*) written in other languages

The java virtual machine (JVM)

Source code → **(Java compiler)** → **Bytecode**
Java program JVM

Bytecode → **(Java interpreter)** → **Machine code**
JVM Real Machine

The first java program

Source: Welcome2Java.java

The first java program

Source: Welcome2Java.java

```
class Welcome2Java{
    public static void main(String args[]){
        System.out.print("Welcome2Java");
    }
}
```

The first java program

Source: Welcome2Java.java

```
class Welcome2Java{
    public static void main(String args[]){
        System.out.print("Welcome2Java");
    }
}
```

Compilation: javac Welcome2Java.java

The first java program

Source: Welcome2Java.java

```
class Welcome2Java{
    public static void main(String args[]){
        System.out.print("Welcome2Java");
    }
}
```

Compilation: javac Welcome2Java.java

"Welcome2Java.java" is compiled and the bytecode "Welcome2Java.class" is created

The first java program

Source: Welcome2Java.java

```
class Welcome2Java{
    public static void main(String args[]){
        System.out.print("Welcome2Java");
    }
}
```

Compilation: javac Welcome2Java.java

"Welcome2Java.java" is compiled and the bytecode "Welcome2Java.class" is created

Execution: java Welcome2Java

The first java program

Source: Welcome2Java.java

```
class Welcome2Java{
    public static void main(String args[]){
        System.out.print("Welcome2Java");
    }
}
```

Compilation: javac Welcome2Java.java

"Welcome2Java.java" is compiled and the bytecode "Welcome2Java.class" is created

Execution: java Welcome2Java

Welcome2Java**(cursor here!!!)**

Dissecting the code

```
import java.*;
class C1{
    statement;
    method1(){  statement;  }
}
class C2{
    main(){ }
}
class C3{
    method2(){  statement;  }
}
```

Dissecting the code

```
import java.*;
class C1{
    statement;
    method1(){  statement;  }
}
class C2{
    main(){ }
}
class C3{
    method2(){  statement;  }
}
```

Note: the program name should be **C2.java**.

Specifying main()

`public static void main()` specifies:

- main method should be unprotected i.e. accessible to all other classes (denoted by the access specifier `public`)
- main method belongs to the entire class and not a part of any objects of the class (denoted by the type `static`)
- main method does not return any value (denoted by the return type `void`)

* Remember that a main method is always `static`.

Specifying main()

`public static void main()` specifies:

- main method should be unprotected i.e. accessible to all other classes (denoted by the access specifier `public`)
- main method belongs to the entire class and not a part of any objects of the class (denoted by the type `static`)
- main method does not return any value (denoted by the return type `void`)

* Remember that a main method is always `static`.

Note: The position of `public` and `static` can be interchanged.

Command line arguments

`String args[]` allows the program to take command line arguments. E.g., the execution with

java Welcome2Java First Year MTech CS

assigns the following:

$$\begin{aligned} \text{args}[0] &\leftarrow \text{First} \\ \text{args}[1] &\leftarrow \text{Year} \\ \text{args}[2] &\leftarrow \text{MTech} \\ \text{args}[3] &\leftarrow \text{CS} \\ \text{args}[4] &\leftarrow \text{NULL} \\ &\vdots \end{aligned}$$

Command line arguments

`String args[]` allows the program to take command line arguments. E.g., the execution with

java Welcome2Java First Year MTech CS

assigns the following:

$$\begin{aligned} \text{args}[0] &\leftarrow \text{First} \\ \text{args}[1] &\leftarrow \text{Year} \\ \text{args}[2] &\leftarrow \text{MTech} \\ \text{args}[3] &\leftarrow \text{CS} \\ \text{args}[4] &\leftarrow \text{NULL} \\ &\vdots \end{aligned}$$

Note: `String args[]` can also be written as `String[] args`.

The comments

Single-line comment:

```
// ...
```

Multi-line comment:

```
/* ...
...
... */
```

Documentation comment:

```
/** ... */
```

The comments

Single-line comment:

```
// ...
```

Multi-line comment:

```
/* ...
...
... */
```

Documentation comment:

```
/** ... */
```

Note: Comments cannot be nested.

Basic characteristics

- A java program (except applets) name should be the class name containing the main method
- There should be at most one class (except for applets having no) that contains the main method
- Every statement must end with a semicolon
- Java is case sensitive

Basic characteristics

- A java program (except applets) name should be the class name containing the main method
- There should be at most one class (except for applets having no) that contains the main method
- Every statement must end with a semicolon
- Java is case sensitive

Note: Applets are obsolete since 2017.

Freeform language

A java statement can be broken into multiple lines. E.g., the statements

```
System.out.print("Welcome2Java");
```

and

```
System.out.print
("Welcome2Java");
```

are both same producing the same output

Strongly typed language

```
char c;
int i;
float f;
```

Here, the data types of c, i, f become fixed. They cannot be used interchangeably.

Strongly typed language

```
char c;
int i;
float f;
```

Here, the data types of c, i, f become fixed. They cannot be used interchangeably. Note that, assigning "i = 2.9" is suicidal.

Strongly typed language

```
char c;
int i;
float f;
```

Here, the data types of c, i, f become fixed. They cannot be used interchangeably. Note that, assigning "i = 2.9" is suicidal.

However, the following conversions are possible with type casting:

`byte` to `short`, `char`, `int`, `long`, `float`, `double`
`short` to `int`, `long`, `float`, `double`
`char` to `int`, `long`, `float`, `double`
`int` to `long`, `float`, `double`
`long` to `float`, `double`
`float` to `double`

E.g., `int i = 10; float f = (float)i;`

Class

Classes are the building blocks of a java program

A class is like a template that describes the behaviors/states that objects of its type support.

Class

Classes are the building blocks of a java program

A class is like a template that describes the behaviors/states that objects of its type support.

Defining a class:

```
class Summation{
    int first, second;
    void Receive(int a, int b){
        first = a; second = b;
    }
    int Add(){
        int result = first + second;
        return(result);
    }
}
```

Object

An object is an instance of a class

Object

An object is an instance of a class

Creating an object:

```
Summation s; // Declaration of object
```

Object

An object is an instance of a class

Creating an object:

```
Summation s; // Declaration of object
s = new Summation();} // Instantiation
```

Object

An object is an instance of a class

Creating an object:

```
Summation s; // Declaration of object
s = new Summation();} // Instantiation
```

Alternatively, you can also write

```
Summation s = new Summation();
```

Object

An object is an instance of a class

Creating an object:

```
Summation s; // Declaration of object
s = new Summation();} // Instantiation
```

Alternatively, you can also write

```
Summation s = new Summation();
```

Accessing class elements with object:

```
s.first; // Accessing the variable 'first'
```

Object

An object is an instance of a class

Creating an object:

```
Summation s; // Declaration of object
s = new Summation();} // Instantiation
```

Alternatively, you can also write

```
Summation s = new Summation();
```

Accessing class elements with object:

```
s.first; // Accessing the variable 'first'
s.Receive(10, 20); // Accessing the method 'Receive()'
```

Object

An object is an instance of a class

Creating an object:

```
Summation s; // Declaration of object
s = new Summation();} // Instantiation
```

Alternatively, you can also write

```
Summation s = new Summation();
```

Accessing class elements with object:

```
s.first; // Accessing the variable 'first'
s.Receive(10, 20); // Accessing the method 'Receive()'
n = s.Add(); // Accessing the method 'Add()'
```

Constructor

Constructor is a method having the same name as the class

Constructor

Constructor is a method having the same name as the class

Defining a constructor:

```
class Summation{
    int first, second;
    Summation(int a, int b){          // No return type
        first = a; second = b;
    }
    int Add(){
        return(first + second);
    }
}
```

Constructor

Constructor is a method having the same name as the class

Defining a constructor:

```
class Summation{
    int first, second;
    Summation(int a, int b){          // No return type
        first = a; second = b;
    }
    int Add(){
        return(first + second);
    }
}
```

Calling a constructor:

```
Summation s = new Summation(6, 28);
```

Immutable classes and objects

Immutable classes are java classes whose objects can not be modified once created, and such objects are known as immutable objects. E.g., `String` is immutable in java.

Any modification in immutable objects results into a new object.

Mostly, the immutable classes are also final in java, in order to prevent subclass from overriding methods in java which can compromise immutability.

Polymorphism

Polymorphism is the ability to take more than one form

Polymorphism

Polymorphism is the ability to take more than one form

```
class Summation{
    int first, second;
    void Receive(int a){          // Function overloading
        first = a;
    }
    void Receive(int a, int b){ // Function overloading
        first = a; second = b;
    }
    int Add(){
        int result = first + second;
        return(result);
    }
}
```

Polymorphism

```
class Summation{
    int first;
    double second;
    void Receive(int a){    // Function overloading
        first = a;
    }
    void Receive(double b){ // Function overloading
        second = b;
    }
}
```

Polymorphism

```
class Summation{
    int first;
    double second;
    void Receive(int a){    // Function overloading
        first = a;
    }
    void Receive(double b){ // Function overloading
        second = b;
    }
}
```

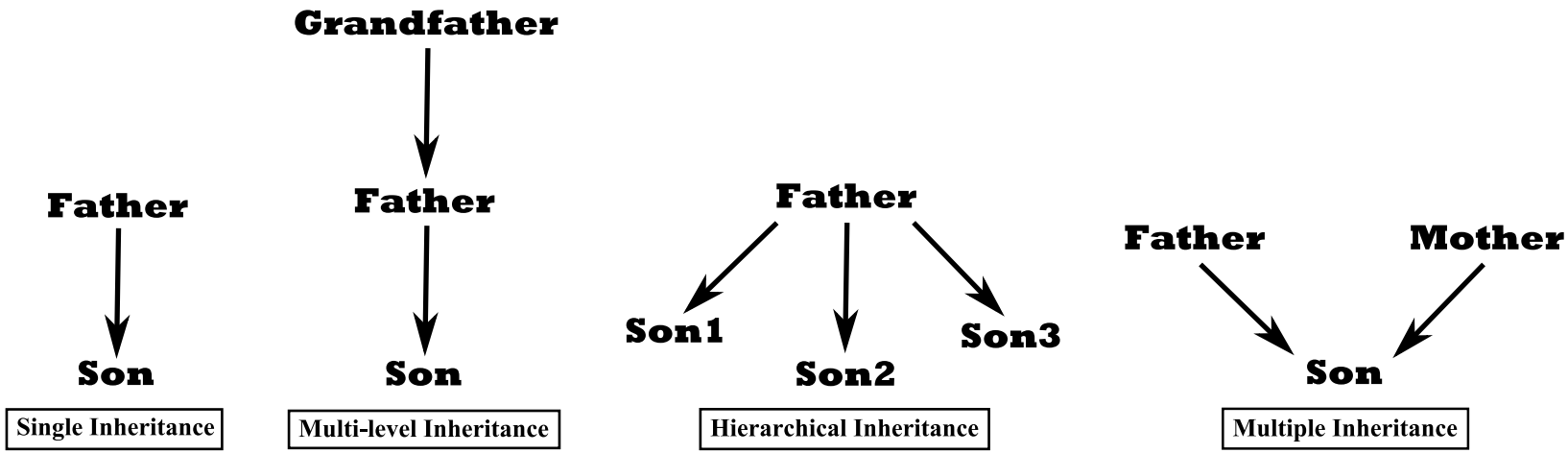
Note: Polymorphism is possible by differentiating the number of arguments or their types but not just by differentiating the return types.

Inheritance

Inheritance means acquiring the properties of another class

Inheritance

Inheritance means acquiring the properties of another class



Note: subclass (child class/derived class) inherits the properties of the superclass (parent class/base class).

Inheritance

Single Inheritance:

```
class Father{
...
}
class Son extends Father{ ... }
```

Inheritance

Single Inheritance:

```
class Father{
...
}
class Son extends Father{ ... }
```

Multi-level Inheritance:

```
class Grandfather{
...
}
class Father extends Grandfather{ ... }
class Son extends Father{ ... }
```

Inheritance

Hierarchical Inheritance:

```
class Father{
...
}
class Son1 extends Father{ ... }
class Son2 extends Father{ ... }
class Son3 extends Father{ ... }
```

Inheritance

Hierarchical Inheritance:

```
class Father{
...
}
class Son1 extends Father{ ... }
class Son2 extends Father{ ... }
class Son3 extends Father{ ... }
```

Multiple inheritance: Not possible in general, but with interfaces

Inheritance

Priority of a subclass is always more than the superclass. This is reflected through *function overriding*.

```
class Father{
    Show(){ System.out.print(‘‘Flip overrides Flop’’); }
}
class Son extends Father{
    Show(){ System.out.print(‘‘Flop overrides Flip’’); }
}
class FunOver{
    public static void main(String args[]){
        Son s = new Son();
        s.Show();
    }
}
```

Inheritance

Priority of a subclass is always more than the superclass. This is reflected through *function overriding*.

```
class Father{
    Show(){ System.out.print(‘‘Flip overrides Flop’’); }
}
class Son extends Father{
    Show(){ System.out.print(‘‘Flop overrides Flip’’); }
}
class FunOver{
    public static void main(String args[]){
        Son s = new Son();
        s.Show();
    }
}
```

Output: `Flop overrides Flip`

Inheritance

Super method:

It is a method that can pass values from a subclass to a superclass and is called as

```
super(x, y)
```

where `x` and `y` are arguments

Inheritance

Super method:

It is a method that can pass values from a subclass to a superclass and is called as

```
super(x, y)
```

where `x` and `y` are arguments

Final class:

It is a special class for which no subclass can be defined is known as a final class and it is declared as

```
final class FClass{ ... }
```

Encapsulation

Encapsulation is a mechanism of wrapping data and methods

Encapsulation

Encapsulation is a mechanism of wrapping data and methods

An object = data + method