Java and Web Technology

Introduction

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

February, 2014

1 Introduction
- What is Java?
- History
- Why java is called java?
- The limitations of java

2 Features of java
- General-purpose
- Platform-independent
- Object-oriented
- Other important features

3 Web technology
- What is web?
- History

4 Suggested Reading
- Resources

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

History

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

1991: Sun Microsystems initiates project led by James Gosling

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

The limitations of java

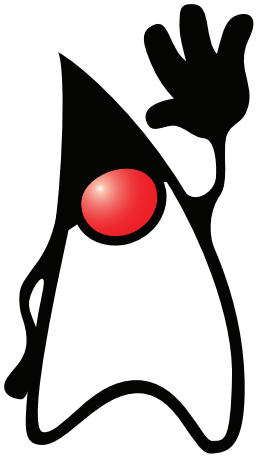
"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

– 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
- Applets cannot stream data directly into the browser

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.

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

What is web?

World wide web (WWW) or simply *web*

What is web?

World wide web (WWW) or simply *web* does not mean the internet!

What is web?

World wide web (WWW) or simply *web* does not mean the internet!

It is a global information medium which users can edit via devices connected to the Internet. So, web is a service that operates over the Internet.

History

1963: Ted Nelson coins the term *HyperText*

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0
1995: Release of Opera 1.0, PHP, IE 1.0 and JavaScript

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0
1995: Release of Opera 1.0, PHP, IE 1.0 and JavaScript
1996: Publishing of Cascading Style Sheets (CSS)

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0
1995: Release of Opera 1.0, PHP, IE 1.0 and JavaScript
1996: Publishing of Cascading Style Sheets (CSS)
1998: Release of XML 1.0 and Mozilla

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0
1995: Release of Opera 1.0, PHP, IE 1.0 and JavaScript
1996: Publishing of Cascading Style Sheets (CSS)
1998: Release of XML 1.0 and Mozilla
2000: Release of XHTML 1.0

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0
1995: Release of Opera 1.0, PHP, IE 1.0 and JavaScript
1996: Publishing of Cascading Style Sheets (CSS)
1998: Release of XML 1.0 and Mozilla
2000: Release of XHTML 1.0
2004: Release of Firefox 1.0 with WYSIWYG code editor

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0
1995: Release of Opera 1.0, PHP, IE 1.0 and JavaScript
1996: Publishing of Cascading Style Sheets (CSS)
1998: Release of XML 1.0 and Mozilla
2000: Release of XHTML 1.0
2004: Release of Firefox 1.0 with WYSIWYG code editor
2006: Release of jQuery

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0
1995: Release of Opera 1.0, PHP, IE 1.0 and JavaScript
1996: Publishing of Cascading Style Sheets (CSS)
1998: Release of XML 1.0 and Mozilla
2000: Release of XHTML 1.0
2004: Release of Firefox 1.0 with WYSIWYG code editor
2006: Release of jQuery
2008: Release of Google Chrome

History

1963: Ted Nelson coins the term *HyperText*
1989: Tim Berners-Lee invents WWW
1990: Tim Berners-Lee designs the first web browser (editor) WorldWideWeb at CERN
1991: *HTML Tags* and HTTP 0.9 specifications arrive
1993: HTML published in full
1994: Release of Python 1.0 and Netscape Navigator 1.0
1995: Release of Opera 1.0, PHP, IE 1.0 and JavaScript
1996: Publishing of Cascading Style Sheets (CSS)
1998: Release of XML 1.0 and Mozilla
2000: Release of XHTML 1.0
2004: Release of Firefox 1.0 with WYSIWYG code editor
2006: Release of jQuery
2008: Release of Google Chrome
More details: http://webdirections.org/history

Resources

Books

1. Herbert Shildt, Java: The Complete Reference, Tata Mcgraw-Hill.
2. Cay S. Horstmann and Gary Cornell, Core Java^{TM} 2, Volume II – Advanced Features, Prentice Hall, 2001.
3. Thomas Powell, Javascript: The Complete Reference, Tata Mcgraw-Hill.
4. James Keogh, J2EE: The Complete Reference, Tata Mcgraw-Hill.
5. Rasmus Lerdorf *et al.*, Programming PHP, O'Reilly.

Journals

1. ACM SIGPLAN Notices.
2. ACM Transactions on Programming Languages and Systems.

Wiki: http://mca402-jwt.wikispaces.com

Java and Web Technology

Java Basics

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

February, 2014

1 Fundamentals of java
- The java virtual machine (JVM)
- The first java program

2 Some characteristics of java

3 OOP concepts in java
- Objects and classes
- Object
- Polymorphism
- Inheritance
- Encapsulation

4 Assignment

The java virtual machine (JVM)

Source code Java program	$\rightarrow$	**(Java compiler)**	$\rightarrow$	**Bytecode** JVM
Bytecode JVM	$\rightarrow$	**(Java interpreter)**	$\rightarrow$	**Machine code** Real Machine

The first java program

<u>Source</u>: 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;
    }
}
```

Dissecting the code

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

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:

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`)

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`)

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`)

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.

Command line arguments

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

java Welcome2Java Second Year MCA KU

assigns the following:

$$\begin{aligned} \text{args}[0] &\leftarrow \text{Second} \\ \text{args}[1] &\leftarrow \text{Year} \\ \text{args}[2] &\leftarrow \text{MCA} \\ \text{args}[3] &\leftarrow \text{KU} \\ \text{args}[4] &\leftarrow \text{NULL} \\ &\vdots \end{aligned}$$

Command line arguments

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

java Welcome2Java Second Year MCA KU

assigns the following:

$$\begin{aligned} \text{args}[0] &\leftarrow \text{Second} \\ \text{args}[1] &\leftarrow \text{Year} \\ \text{args}[2] &\leftarrow \text{MCA} \\ \text{args}[3] &\leftarrow \text{KU} \\ \text{args}[4] &\leftarrow \text{NULL} \\ &\vdots \end{aligned}$$

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

The comments

Single-line comment:

// ...

The comments

Single-line comment:

// ...

Multi-line comment:

/* ...

...

... */

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

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

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

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

Freeform language

A java statement can be broken into multiple lines.

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;
```

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. E.g., assigning "b = 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. E.g., assigning "b = 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`

Strongly typed language

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

Here, the data types of c, i, f become fixed. They cannot be used interchangeably. E.g., assigning "b = 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

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; // Access 'first' from the class '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;` // Access 'first' from the class 'Summation'

`s.Receive(10, 20);` // Access 'Receive()' from the class 'Summation'

Constructor

A constructor is a method having the name same as the class name

Constructor

A constructor is a method having the name same as the class name

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

A constructor is a method having the name same as the class name

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);
```

Polymorphism

Polymorphism (also known as function overloading) is the ability to take more than one form

Polymorphism

Polymorphism (also known as function overloading) is the ability to take more than one form

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

Polymorphism

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

Polymorphism

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

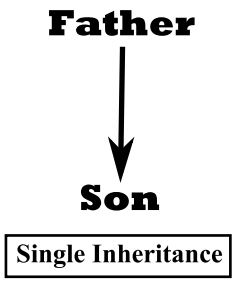
Note: Polymorphism is possible by differentiating the number of arguments or their types but not just by differentiating the return types.

Inheritance

Inheritance means acquiring the properties of another class

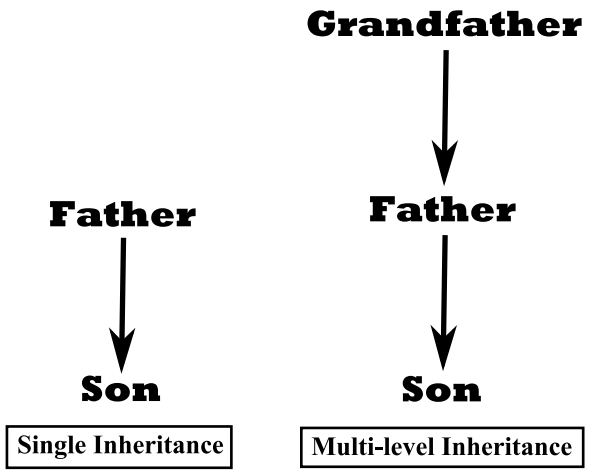
Inheritance

Inheritance means acquiring the properties of another class



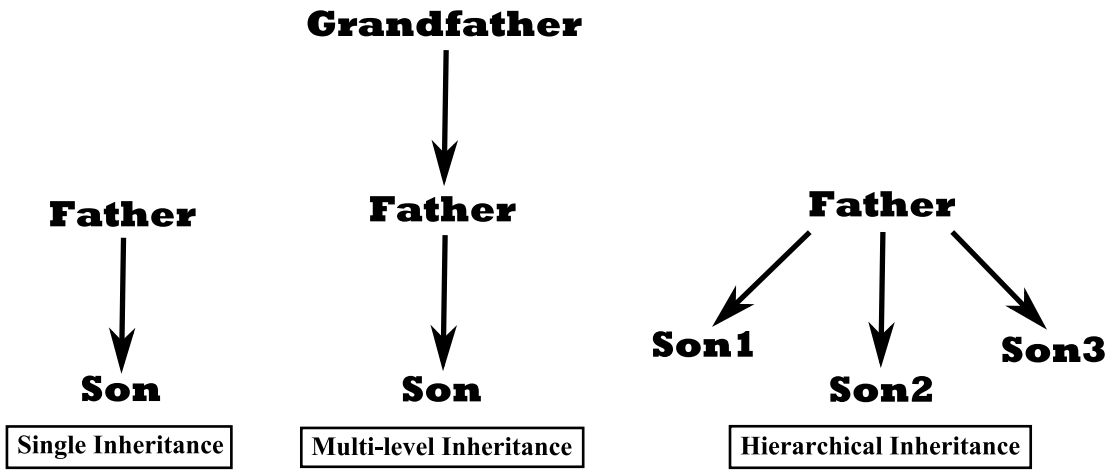
Inheritance

Inheritance means acquiring the properties of another class



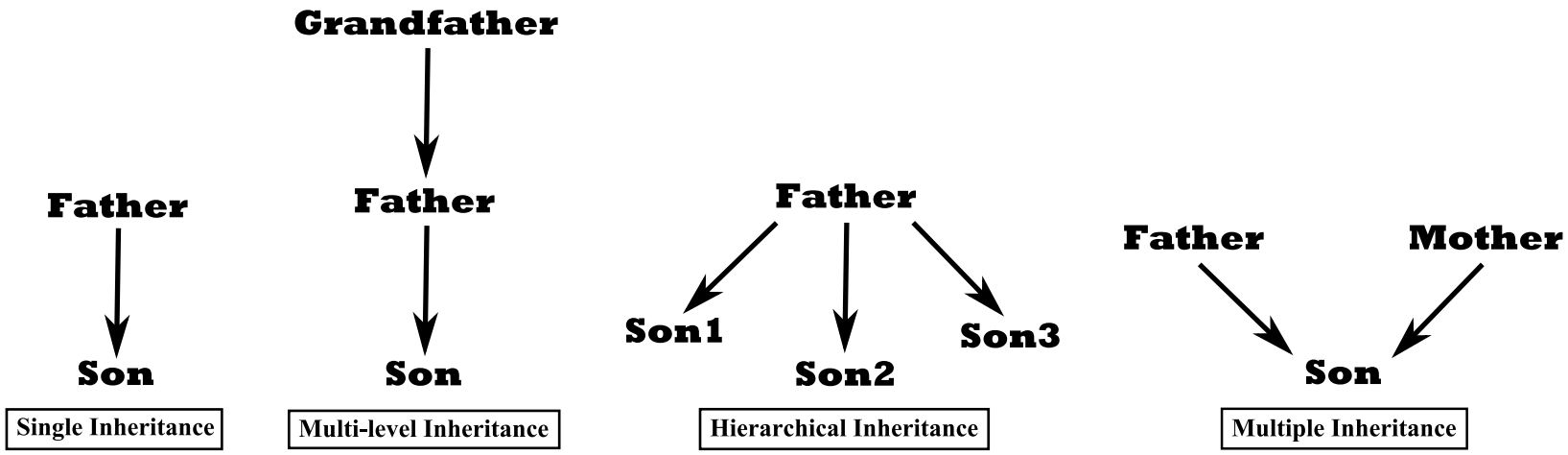
Inheritance

Inheritance means acquiring the properties of another class



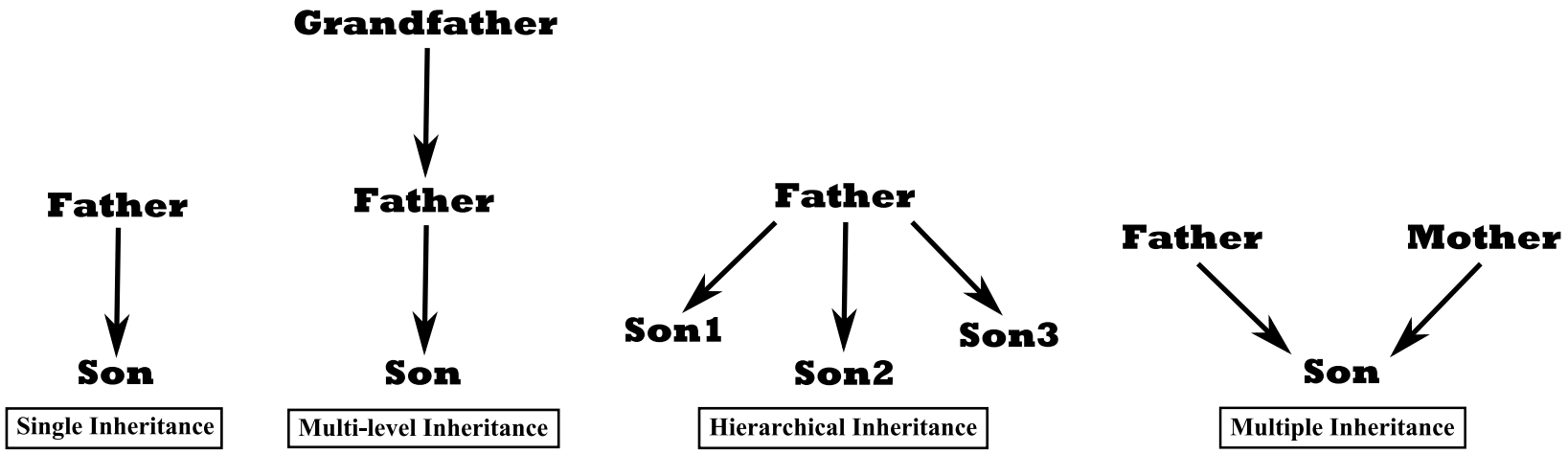
Inheritance

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

The output will be: Flop

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

An object = data + method

Assignment

1. Are the documentation comments in java multi-line?
2. Write the smallest executable java code.
3. Write a single program in java that contains all types of inheritances.

Java and Web Technology

Java I/O, Operators, Control Flow, Arrays and Exceptions

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

February, 2014

1 Java I/O
- Taking inputs
- Showing outputs

2 Operators

3 Branching

4 Loops

5 Arrays
- One dimensional array
- Two dimensional array
- Variable size array

6 Strings

7 Vectors

8 Exception handling

9 Assignment

Taking inputs (with Scanner class)

```
  import java.io.*;
import java.util.Scanner;
class Input{
    public static void main(String args[]) throws IOException{
        int i;
        float f;
        Scanner sc = new Scanner(System.in);
        System.out.print("Age:  ");
        i = sc.nextInt();
        if(i < 0)
            System.out.print("Invalid age!!!");
        else{
            System.out.print("Valid age.  Now enter height:  ");
            f = sc.nextFloat();
            System.out.print("Height = "+f);
        }
    }
}
```

Taking inputs (with DataInputStream class)

```
  import java.io.*;
class Input{
    public static void main(String args[]) throws IOException{
        int i;
        double f;
        boolean c;
        DataInputStream dis = new DataInputStream(System.in);
        System.out.print("Age, height & code:  ");
        i = dis.readInt();
        f = dis.readDouble();
        c = dis.readBoolean();
        System.out.print("Age = " + i + ", height = " + f + "
feet" + " & code = " + c);
    }
}
```

Taking inputs (with DataInputStream class - alternatively)

```
  import java.io.*;
class Input{
    public static void main(String args[]) throws IOException{
        int i;
        float f;
        char c;
        DataInputStream dis = new DataInputStream(System.in);
        System.out.print("Age, height & code:  ");
        i = Integer.parseInt(dis.readLine());
        f = Float.valueOf(dis.readLine()).
floatValue();
        c = (char) System.in.read();
        System.out.print("Age = " + i + ", height = " + f + "
feet" + " & code = " + c);
    }
}
```

Showing outputs

```
System.out.println("String" + variables);
```

Showing outputs

```
System.out.println("String" + variables);
```

```
class Input{
    public static void main(String args[]){
        int i=10;
        // The following two statements are same
        System.out.println("Value of i = "+i);
        System.out.print("Value of i = "+i+"\n");
    }
}
```

Showing outputs

```
System.out.println("String" + variables);
```

```
class Input{
    public static void main(String args[]){
        int i=10;
        // The following two statements are same
        System.out.println("Value of i = "+i);
        System.out.print("Value of i = "+i+"\n");
    }
}
```

Note: Java being strongly typed the format specifiers are not required.

Arithmetic operators

Summation $(+)$

Subtraction $(-)$

Multiplication $(*)$

Division $(/)$

Modulo division $(\%)$

Arithmetic operators

Summation $(+)$

Subtraction $(-)$

Multiplication $(*)$

Division $(/)$

Modulo division $(\%)$

Note: Modulo devision operator also work on floats

Relational operators

Less than $(<)$

Less than equals to $(<=)$

Greater than $(>)$

Greater than equals to $(>=)$

Equals to $(==)$

Not equals to $(!=)$

Logical operators

Logical and (&&)

Logical or (||)

Logical not (!)

Assignment operators

Assignment $(=)$

Increment and decrement operators

Increment $(++)$

Decrement $(--)$

Conditional operators

Ternary (? :)

```
 int i = 0;
int j = 1;
int result = (i > j)?  i :  j;
```

Bitwise operators

Bitwise and (&)

Bitwise or (|)

Bitwise not (~)

Bitwise xor (∧)

Left shift (<<)

Right shift (>>)

Right shift with zero padding (>>>)

Special operators

Dot (.)

```
class Input{
    public static void main(String args[]){
        Summation s = new Summation();
        s.Add();
    }
}
```

if-else

```
 if(Condition){
    statement 1;
    statement 2;
}
else{
    statement 3; // Execute if Condition fails
}
```

if-else

```
 if(Condition){
    statement 1;
    statement 2;
}
else{
    statement 3; // Execute if Condition fails
}
```

Note: The 'Condition' should be in `Boolean` form.

for loop

```
for(Initialization;Condition;Increment/decrement){
    statement 1;
    statement 2;
}
```

for loop

```
for(Initialization;Condition;Increment/decrement){
    statement 1;
    statement 2;
}
```

Alternatively, we can write

```
for(Initialization;Condition;Increment/decrement)
    statement 3;
```

for loop

```
for(Initialization;Condition;Increment/decrement){
    statement 1;
    statement 2;
}
```

Alternatively, we can write

```
for(Initialization;Condition;Increment/decrement)
    statement 3;
```

Note: The 'Condition' should be in `Boolean` form.

while loop

```
while(Condition){
    statement 1;
    statement 2;
}
```

while loop

```
while(Condition){
    statement 1;
    statement 2;
}
```

Alternatively, we can write

```
while(Condition)
    statement 3;
```

while loop

```
while(Condition){
    statement 1;
    statement 2;
}
```

Alternatively, we can write

```
while(Condition)
    statement 3;
```

Note: The 'Condition' should be in `Boolean` form.

do-while loop

```
do{
    statement 1;
    statement 2;
}while(Condition)
```

do-while loop

```
do{
    statement 1;
    statement 2;
}while(Condition)
```

Note: The 'Condition' should be in `Boolean` form.

One dimensional array

Creating an array:

```
class Array{
    public static void main(String args[]){
        int semester[]; // Declaration
        semester = new int[6]; // Memory allocation
    }
}
```

One dimensional array

Creating an array:

```
class Array{
    public static void main(String args[]){
        int semester[]; // Declaration
        semester = new int[6]; // Memory allocation
    }
}
```

The declaration and memory allocation can be combined together as follows:

```
int semester[] = new int[6];
```

One dimensional array

Creating an array:

```
class Array{
    public static void main(String args[]){
        int semester[]; // Declaration
        semester = new int[6]; // Memory allocation
    }
}
```

The declaration and memory allocation can be combined together as follows:

```
int semester[] = new int[6];
```

Note: The declaration can also be like "`int[] semester;`"

One dimensional array

Initializing an array:

```
class Array{
    public static void main(String args[]){
        // Initialization while declaring
        int semester[] = {1,2,3,4,5,6};
    }
}
```

One dimensional array

Initializing an array:

```
class Array{
    public static void main(String args[]){
        // Initialization while declaring
        int semester[] = {1,2,3,4,5,6};
    }
}
```

Note: The length of the array can be returned with "`semester.length;`"

Two dimensional array

Creating an array:

```
class Array{
  public static void main(String args[]){
      // Declaration and allocation
      float result[][] = new float[6][30];
  }
}
```

Variable size array

Creating an array of variable sizes:

```
class Array{
  public static void main(String args[]){
      char result[][] = new char[5][]; // Declaring number of rows
```

Variable size array

Creating an array of variable sizes:

```
class Array{
  public static void main(String args[]){
      char result[][] = new char[5][]; // Declaring number of rows
      result[0] = new char[10]; // Row 1 have 10 columns
      result[1] = new char[20]; // Row 2 have 20 columns
      result[2] = new char[30]; // Row 3 have 30 columns
      result[3] = new char[40]; // Row 4 have 40 columns
      result[4] = new char[50]; // Row 5 have 50 columns
  }
}
```

Strings

Creating a string:

```
class ABCD{
    public static void main(String args[]){
        String name; // Declaration
        name = new String("Y-"); // Initialization
    }
}
```

Strings

Creating a string:

```
class ABCD{
    public static void main(String args[]){
        String name; // Declaration
        name = new String("Y-"); // Initialization
    }
}
```

The declaration and initialization can be combined together as follows:

```
String name = new String("Y-");
```

Strings

Creating an array of strings:

```
class StrArray{
    public static void main(String args[]){
        // Declaration
        String name[] = new String[10];
    }
}
```

Strings

Creating an array of strings:

```
class StrArray{
    public static void main(String args[]){
        // Declaration
        String name[] = new String[10];
    }
}
```

Note: Java strings can be concatenated with the '+' operator

Strings

Creating an array of strings:

```
class StrArray{
    public static void main(String args[]){
        // Declaration
        String name[] = new String[10];
    }
}
```

Note: Java strings can be concatenated with the '+' operator E.g., "`String name = "Life" + "of" + "Pi";`" will result "LifeofPi".

Vectors

The `Vector` class implements a growable array of objects. However, its size can grow/shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Vectors

The `Vector` class implements a growable array of objects. However, its size can grow/shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Creating a vector:

```
class V{
    public static void main(String args[]){
        Vector obj1 = new Vector(); // Without size
        Vector obj2 = new Vector(10); // With size
    }
}
```

Vectors

The `Vector` class implements a growable array of objects. However, its size can grow/shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Creating a vector:

```
class V{
    public static void main(String args[]){
        Vector obj1 = new Vector(); // Without size
        Vector obj2 = new Vector(10); // With size
    }
}
```

Similar functionality can also be obtained with the `ArrayList` class.

Vectors

The `Vector` class implements a growable array of objects. However, its size can grow/shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Creating a vector:

```
class V{
    public static void main(String args[]){
        Vector obj1 = new Vector(); // Without size
        Vector obj2 = new Vector(10); // With size
    }
}
```

Similar functionality can also be obtained with the `ArrayList` class.

Note: While using vectors you should include "import `java.util.*`"

Differentiating the Vector class from ArrayList

Vector	ArrayList
1. Vectors are synchronized.	1. ArrayList is unsynchronized.
2. Any method that touches the Vector's contents is thread safe.	2. ArrayList is not thread safe.
3. Data growth can be efficiently handled with Vectors.	3. Data growth is not efficient to handle with ArrayList.

Exception handling

```
try{
    statement 1;
    statement 2;
}
```

Exception handling

```
 try{
    statement 1;
    statement 2;
}
catch(Exception type 1){
    statement 3; // Executed for exception type 1
}
```

Exception handling

```
 try{
    statement 1;
    statement 2;
}
catch(Exception type 1){
    statement 3; // Executed for exception type 1
}
catch(Exception type 2){
    statement 4; // Executed for exception type 2
}
```

Exception handling

```
 try{
    statement 1;
    statement 2;
}
catch(Exception type 1){
    statement 3; // Executed for exception type 1
}
catch(Exception type 2){
    statement 4; // Executed for exception type 2
}
```

Note: A `try` block should be matched with at least one `catch` or `finally` block.

Use of `finally` in exception handling

A `finally` block can also be matched with a `try` block to handle exceptions. Once entered, the `finally` block will always execute no matter if you return from it or an exception is thrown within the try block.

Use of `finally` in exception handling

A `finally` block can also be matched with a `try` block to handle exceptions. Once entered, the `finally` block will always execute no matter if you return from it or an exception is thrown within the try block.

Only if you call `exit()` before the `finally` is reached, it will skip the execution.

An example

```
 import java.io.DataInputStream;
class Input{
    public static void main(String args[]){
        int i;
        DataInputStream in = new
DataInputStream(System.in);
        System.out.println("Enter the value = ");
        try{
            i = Integer.parseInt(in.readLine());
        }
        catch(Exception e){ // 'e' can be renamed
            System.out.print(e);
        }
    }
}
```

catch exception types

- **IOException:** Failure in taking input or showing output

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)
- **ArrayIndexOutOfBoundsException:** Array index out of bounds

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)
- **ArrayIndexOutOfBoundsException:** Array index out of bounds
- **StringIndexOutOfBoundsException:** String index out of bounds

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)
- **ArrayIndexOutOfBoundsException:** Array index out of bounds
- **StringIndexOutOfBoundsException:** String index out of bounds
- **NegativeArraySizeException:** Negative array size error

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)
- **ArrayIndexOutOfBoundsException:** Array index out of bounds
- **StringIndexOutOfBoundsException:** String index out of bounds
- **NegativeArraySizeException:** Negative array size error
- **NullPointerException:** Referencing a null object

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)
- **ArrayIndexOutOfBoundsException:** Array index out of bounds
- **StringIndexOutOfBoundsException:** String index out of bounds
- **NegativeArraySizeException:** Negative array size error
- **NullPointerException:** Referencing a null object
- **NumberFormatException:** Failure of string to number conversion

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)
- **ArrayIndexOutOfBoundsException:** Array index out of bounds
- **StringIndexOutOfBoundsException:** String index out of bounds
- **NegativeArraySizeException:** Negative array size error
- **NullPointerException:** Referencing a null object
- **NumberFormatException:** Failure of string to number conversion
- **OutOfMemoryException:** Not enough memory

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)
- **ArrayIndexOutOfBoundsException:** Array index out of bounds
- **StringIndexOutOfBoundsException:** String index out of bounds
- **NegativeArraySizeException:** Negative array size error
- **NullPointerException:** Referencing a null object
- **NumberFormatException:** Failure of string to number conversion
- **OutOfMemoryException:** Not enough memory
- **SecurityException:** Applet causes security problems

catch exception types

- **IOException:** Failure in taking input or showing output
- **ArithmeticException:** Arithmetic errors (e.g., divide by zero)
- **ArrayIndexOutOfBoundsException:** Array index out of bounds
- **StringIndexOutOfBoundsException:** String index out of bounds
- **NegativeArraySizeException:** Negative array size error
- **NullPointerException:** Referencing a null object
- **NumberFormatException:** Failure of string to number conversion
- **OutOfMemoryException:** Not enough memory
- **SecurityException:** Applet causes security problems

... Find more in books.

Throwing exception for a single statement

```
 import java.io.DataInputStream;
class Input{
    public static void main(String args[]){
        int i;
        DataInputStream in = new
DataInputStream(System.in);
        System.out.println("Enter the value = ");
        // No try-catch
        i = Integer.parseInt(in.readLine());
        if(i == 0){
            throw new IOException("Unable to
operate");
        }
        System.out.print(i);
    }
}
```

Throwing exception for an entire method

```
 import java.io.DataInputStream;
class Input{
    public static void main(String args[]) throws
IOException{
        int i;
        DataInputStream in = new
DataInputStream(System.in);
        System.out.println("Enter the value = ");
        // No try-catch
        i = Integer.parseInt(in.readLine());
        System.out.print(i);
    }
}
```

Nested try-catch

```
 try{
    statement 1;
    try{ // Nested try
        statement 2;
    }
    catch(Exception type 1){ ...  }
}
catch(Exception type 2){
    try{
        statement 4;
    }
    catch(Exception type 3){
        statement 5;
    }
}
```

Assignment

1. What are those operators not common between C and java?
2. Write a user input program in java to show one sided and two-sided Pascal's triangles using polymorphism.
3. How would you compute the length of string in java?
4. How would you print the exception message in a java program on the output screen?

Java and Web Technology

Built-in Classes, Methods in Java and File Programming

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

February, 2014

1 Introduction

2 Concept of packages

3 Access types of java classes

4 File programming
- Concept of streams
- File object
- File writing
- File reading
- File reading/writing

5 Other controls in file

6 Assignment

Introduction

The power of java lies in its reuse

Introduction

The power of java lies in its reuse

The classes and methods in a java program can be reused

Introduction

The power of java lies in its reuse

The classes and methods in a java program can be reused
– by extending classes and implementing interfaces

Introduction

The power of java lies in its reuse

The classes and methods in a java program can be reused – by extending classes and implementing interfaces

But what is beyond that?

Introduction

The power of java lies in its reuse

The classes and methods in a java program can be reused – by extending classes and implementing interfaces

But what is beyond that?

One can reuse the built-in classes and methods in java

Introduction

The power of java lies in its reuse

The classes and methods in a java program can be reused – by extending classes and implementing interfaces

But what is beyond that?

One can reuse the built-in classes and methods in java

How?

Introduction

The power of java lies in its reuse

The classes and methods in a java program can be reused – by extending classes and implementing interfaces

But what is beyond that?

One can reuse the built-in classes and methods in java

How?

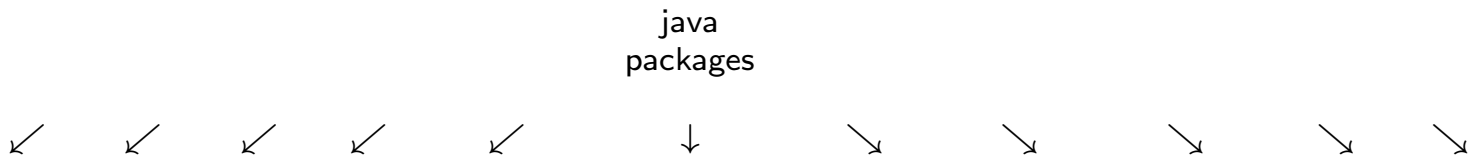
With packages

Basics

A package is a group of variety of classes and/or interfaces taken together

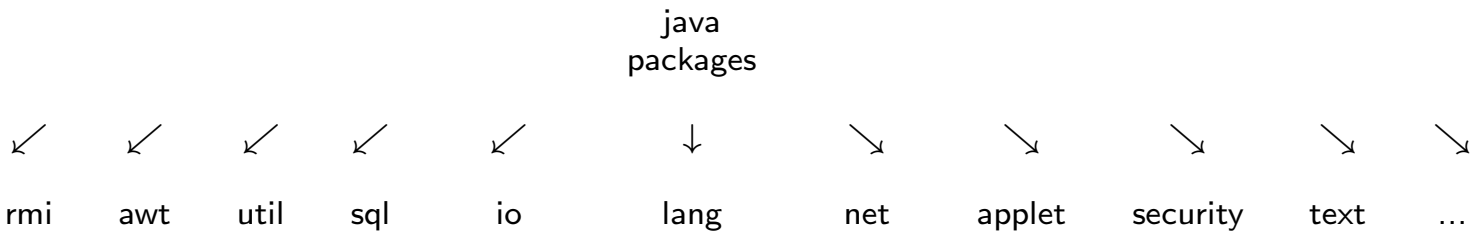
Basics

A package is a group of variety of classes and/or interfaces taken together



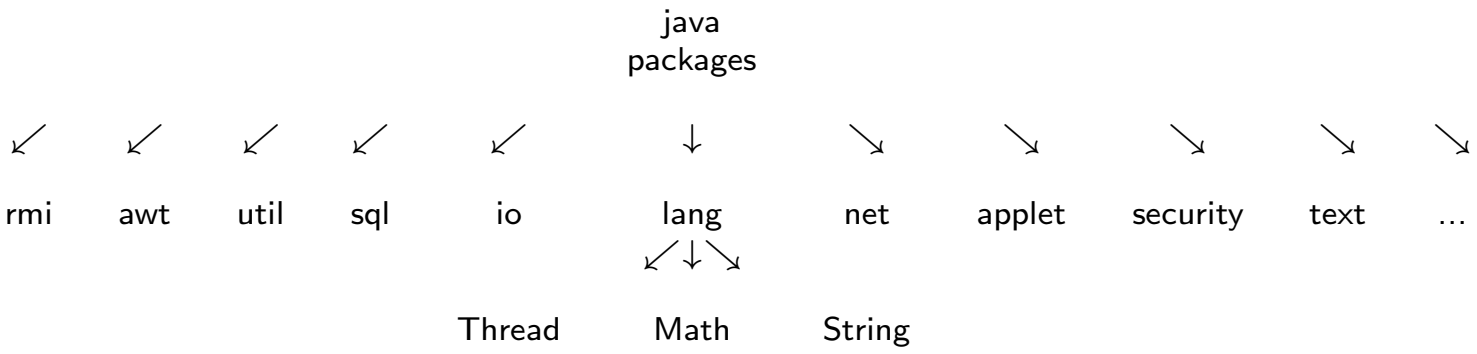
Basics

A package is a group of variety of classes and/or interfaces taken together



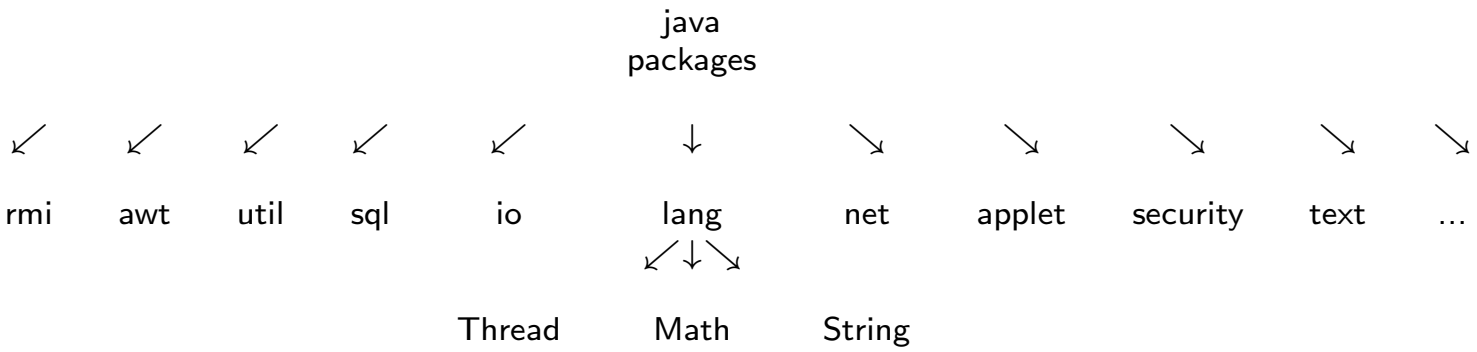
Basics

A package is a group of variety of classes and/or interfaces taken together



Basics

A package is a group of variety of classes and/or interfaces taken together



<u>Note</u>: Find more in books

Organization and use of a package

java . lang . Math . sqrt()

Organization and use of a package

java	.	lang	.	Math	.	sqrt()
		↓		↓		↓
		package		class		method

Organization and use of a package

java	.	lang	.	Math	.	sqrt()
		↓		↓		↓
		package		class		method

```
 import java.lang.Math; // Alternatively java.lang.*
class SquareRoot{
    public static void main(String [] args){
        double pi = 3.1415926535, value;
        value = Math.sqrt(pi); // Square root of pi
        System.out.printl("Square root of pi:
"+value);
    }
}
```

Organization and use of a package

java	.	lang	.	Math	.	sqrt()
		↓		↓		↓
		package		class		method

```
 import java.lang.Math; // Alternatively java.lang.*
class SquareRoot{
    public static void main(String [] args){
        double pi = 3.1415926535, value;
        value = Math.sqrt(pi); // Square root of pi
        System.out.printl("Square root of pi:
"+value);
    }
}
```

Note: Appropriate packages should be imported while reusing classes and methods

Some packages

- **rmi:** For setting remote connections like remote API

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.
- **util:** For utility classes like vectors, hash tables, random numbers, etc.

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.
- **util:** For utility classes like vectors, hash tables, random numbers, etc.
- **sql:** For database connection like with sql

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.
- **util:** For utility classes like vectors, hash tables, random numbers, etc.
- **sql:** For database connection like with sql
- **io:** For input/output operations

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.
- **util:** For utility classes like vectors, hash tables, random numbers, etc.
- **sql:** For database connection like with sql
- **io:** For input/output operations
- **lang:** For supports like strings, mathematical functions, threads, etc.

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.
- **util:** For utility classes like vectors, hash tables, random numbers, etc.
- **sql:** For database connection like with sql
- **io:** For input/output operations
- **lang:** For supports like strings, mathematical functions, threads, etc.
- **net:** For network connections like sockets, URL access, etc.

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.
- **util:** For utility classes like vectors, hash tables, random numbers, etc.
- **sql:** For database connection like with sql
- **io:** For input/output operations
- **lang:** For supports like strings, mathematical functions, threads, etc.
- **net:** For network connections like sockets, URL access, etc.
- **applet:** For implementing applets

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.
- **util:** For utility classes like vectors, hash tables, random numbers, etc.
- **sql:** For database connection like with sql
- **io:** For input/output operations
- **lang:** For supports like strings, mathematical functions, threads, etc.
- **net:** For network connections like sockets, URL access, etc.
- **applet:** For implementing applets
- **security:** For security assignments

Some packages

- **rmi:** For setting remote connections like remote API
- **awt:** For graphical user interfaces like buttons, menus, etc.
- **util:** For utility classes like vectors, hash tables, random numbers, etc.
- **sql:** For database connection like with sql
- **io:** For input/output operations
- **lang:** For supports like strings, mathematical functions, threads, etc.
- **net:** For network connections like sockets, URL access, etc.
- **applet:** For implementing applets
- **security:** For security assignments
- **text:** For handling texts, dates, numbers, etc.

Creating a package

```
 package myPack; // Declaration
public class MyClass{
    char c;
    public void show(){
        int i;
        System.out.println("I am a package ...");
    }
}
class AnotherClass{
    public void Nothing(){
    }
}
```

Creating a package

```
 package myPack; // Declaration
public class MyClass{
    char c;
    public void show(){
        int i;
        System.out.println("I am a package ...");
    }
}
class AnotherClass{
    public void Nothing(){
    }
}
```

Note: This is a .java file

Creating a package

Points to remember:

- The package declaration should be the first line in your source code

Creating a package

Points to remember:

- The package declaration should be the first line in your source code
- There should be one and only one public class in a package file and that class name should be the file name with extension .java

Creating a package

Points to remember:

- The package declaration should be the first line in your source code
- There should be one and only one public class in a package file and that class name should be the file name with extension .java
- Create a subdirectory under the main directory and put the .java file there

Creating a package

Points to remember:

- The package declaration should be the first line in your source code
- There should be one and only one public class in a package file and that class name should be the file name with extension .java
- Create a subdirectory under the main directory and put the .java file there
- Compile it and keep the class file there

Using a package

```
 import myPack.MyClass;
class TestMyPackage{
    public static void main(String [] args){
        MyClass mc = new MyClass();
        mc.show();
    }
}
```

Interesting notes

- Classes are unique to a package but not in other packages

Interesting notes

- Classes are unique to a package but not in other packages
- Package names conventionally start with a lowercase letter

Interesting notes

- Classes are unique to a package but not in other packages
- Package names conventionally start with a lowercase letter
- Class names conventionally start with an uppercase letter

Access types of java classes

	public	**protected**	**friendly (default)**	**private protected**	**private**
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

Access types of java classes

	public	**protected**	**friendly (default)**	**private protected**	**private**
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

Note: The `public`, `protected`, `friendly`, `private protected` and `private` are known as access specifiers.

Concept of streams

Input from →	Medium		Medium	→ **Outputs to**
Mouse				Console
Keyboard	Input		Output	Printer
Primary storage		Java		Primary storage
Secondary storage	stream		stream	Secondary storage
Network				Network

The stream classes in java

Java streams

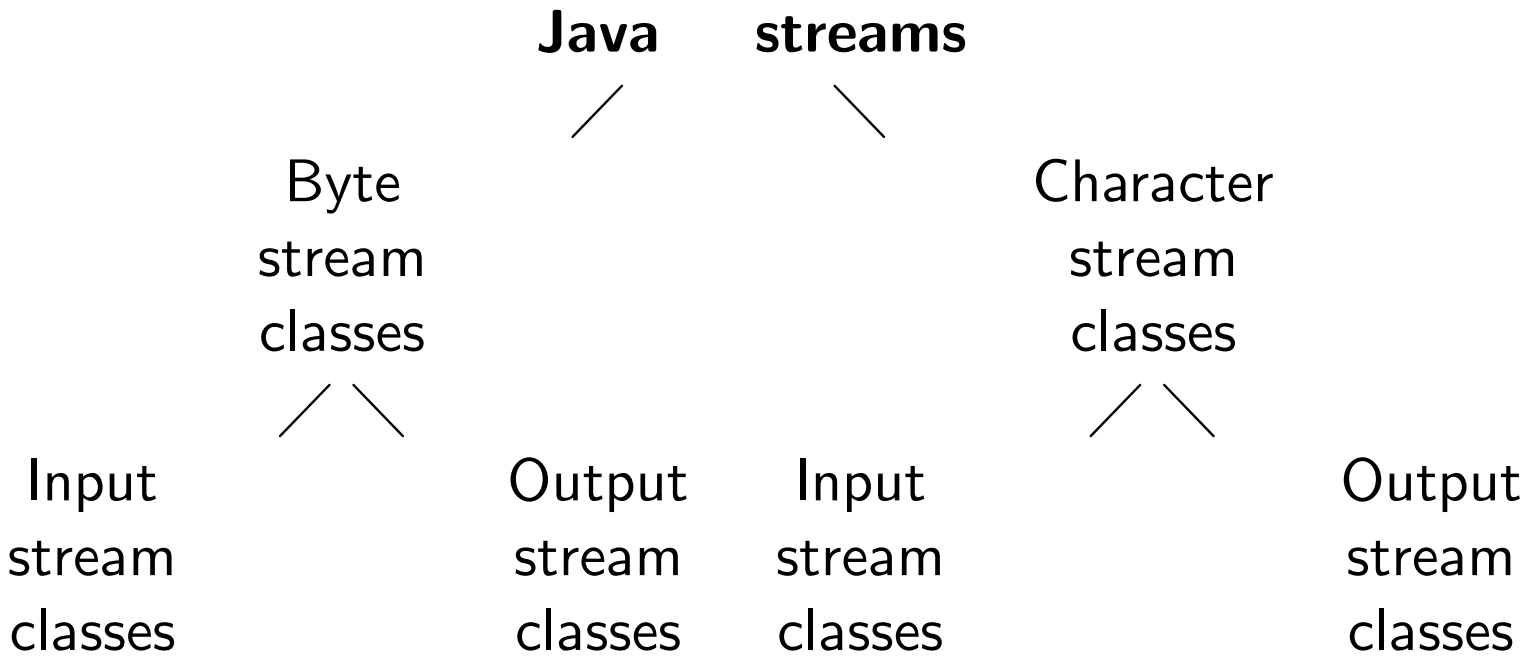
The stream classes in java

Java streams

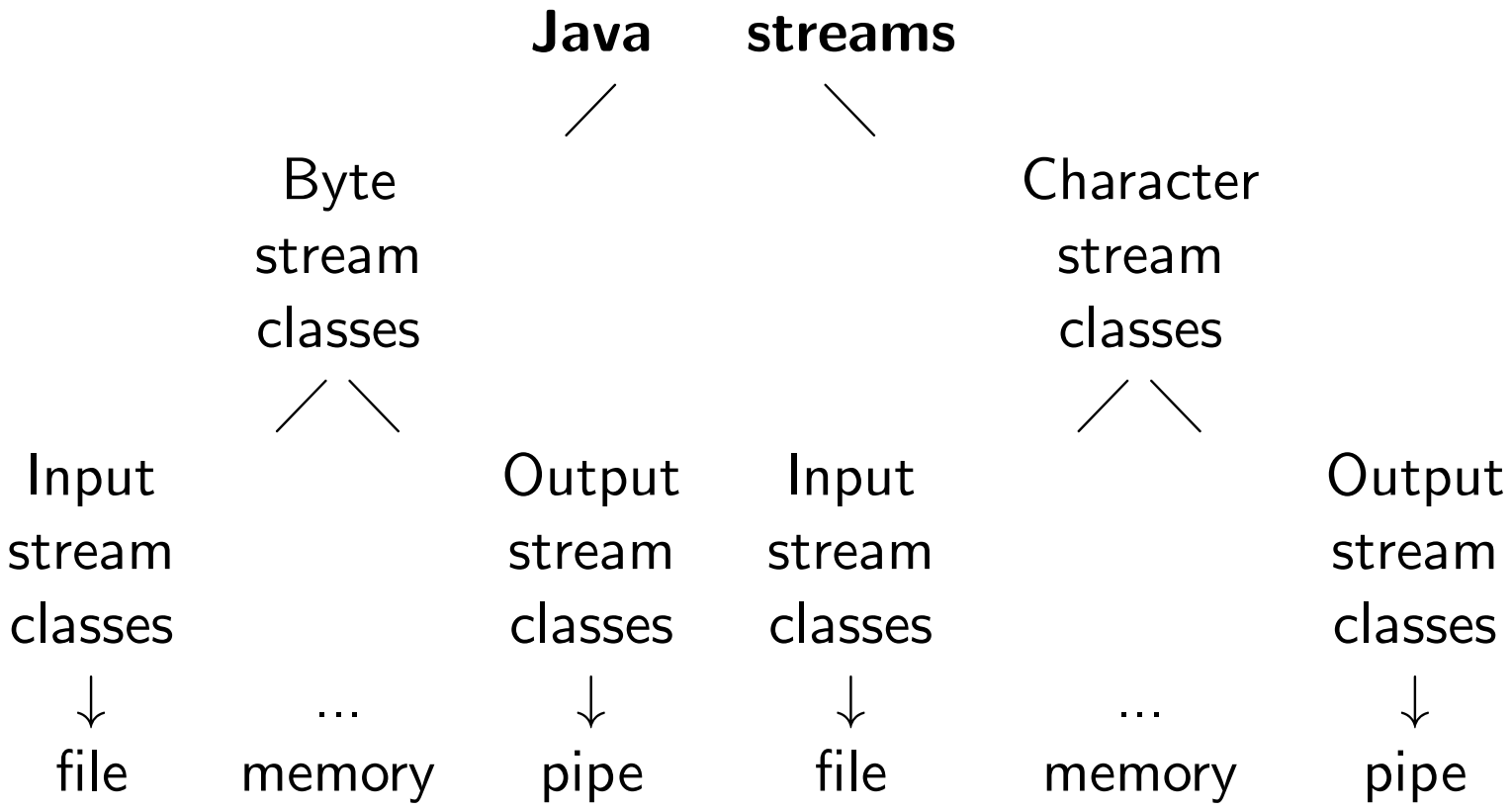
Byte stream classes

Character stream classes

The stream classes in java



The stream classes in java



Creating a file object (directly)

```
class FileTest{
    public static void main(String args[]){
        // Declaration
        FileInputStream fis;
        try{
            // Allocation
            fis = new FileInputStream("Input.txt");
        }
        catch(IOException e){ }
    }
}
```

Creating a file object (directly)

```
class FileTest{
    public static void main(String args[]){
        // Declaration
        FileInputStream fis;
        try{
            // Allocation
            fis = new FileInputStream("Input.txt");
        }
        catch(IOException e){ }
    }
}
```

The declaration and allocation can be combined together as follows:

```
FileInputStream fis = new
FileInputStream("Input.txt");
```

Creating a file object (indirectly)

```
class FileTest{
    public static void main(String args[]){
        // Declaration & allocation (to open file)
        File f = new File("Input.txt");
        // Declaration
        FileInputStream fis;
        try{
            // Indirect allocation
            fis = new FileInputStream(f);
        }
        catch(IOException e){ }
    }
}
```

Writing to a file (using FileOutputStream class)

```
 import java.io.*;
class FileWriteByte{
    public static void main(String args[]){
        byte Players[] =
{'M','e','s','s','i','\t','R','o','n','a','l','d','o'};
        FileOutputStream fos = null;
        try{
            fos = new FileOutputStream("Output.txt");
            int i;
            fos.write(Players);
            fos.close();
        }
        catch(IOException e){ }
    }
}
```

Writing to a file (handling primitive data types)

```
 import java.io.*;
class FileWrite{
    public static void main(String args[]) throws
IOException{
        File f = new File("Input.txt");
        FileOutputStream fos = new
FileOutputStream(f);
        DataOutputStream dos = new
DataOutputStream(fos);
        dos.writeInt(12321);
        dos.writeDouble(12.3);
        dos.writeChar('?');
        dos.close();
        fos.close();
    }
}
```

Writing to a file (using FileWriter class)

```
 import java.io.*;
class FileWrite{
    public static void main(String args[]){
        File f = new File("Output.txt");
        FileWriter fw; // To write in file
        try{
            fw = new FileWriter(f);
            int i = 0;
            while(args[0][i] != '\0'){
                fw.write((char)args[0][i++]);
            }
            fw.close();
        }
        catch(IOException e){ }
    }
}
```

Reading from a file (using FileInputStream class)

```
 import java.io.*;
class FileReadInt{
    public static void main(String args[]){
        FileInputStream fis = null;
        try{
            fis = new FileInputStream("Input.txt");
            int i;
            // The read() method returns an integer
            while((i = fis.read()) != -1){
                System.out.print(i);
            }
            fis.close();
        }
        catch(IOException e){ }
    }
}
```

Reading from a file (handling primitive data types)

```
 import java.io.*;
class FileRead{
    public static void main(String args[]) throws
IOException{
        File f = new File("Input.txt");
        FileInputStream fis = new FileInputStream(f);
        DataInputStream dis = new
DataInputStream(fis);
        System.out.println("Read items:
"+dis.readChar()+", "+dis.readInt()+" and
"+dis.readFloat());
        dis.close();
        fis.close();
    }
}
```

Reading from a file (handling primitive data types)

```
 import java.io.*;
class FileRead{
    public static void main(String args[]) throws
IOException{
        File f = new File("Input.txt");
        FileInputStream fis = new FileInputStream(f);
        DataInputStream dis = new
DataInputStream(fis);
        System.out.println("Read items:
"+dis.readChar()+", "+dis.readInt()+" and
"+dis.readFloat());
        dis.close();
        fis.close();
    }
}
```

Note: The file contents have to be written programmatically.

Reading from a file (using FileReader class)

```
 import java.io.*;
class FileReadChar{
    public static void main(String args[]){
        File f = new File("Input.txt");
        FileReader fr; // To read file
        try{
            fr = new FileReader(f);
            int i;
            while((i = fr.read()) != -1){
                System.out.print((char)i);
            }
            fr.close();
        }
        catch(IOException e){ }
    }
}
```

Reading/writing a file (using RandomAccessFile class)

```
 import java.io.*;
class FileReadAll{
    public static void main(String args[]) throws
IOException{
        RandomAccessFile rafi = new
RandomAccessFile("Input.txt","r");
        RandomAccessFile rafo = new
RandomAccessFile("Output.txt","rw");
        int i = rafi.readInt();
        rafo.writeInt(i);
        System.out.println("DONE!");
        rafi.close();
        rafo.close();
    }
}
```

Other methods in RandomAccessFile class

Read method	Usage	Write method	Usage
readBoolean()		writeBoolean()	
readChar()		writeChar()	
readByte()		writeByte()	
readFloat()		writeFloat()	
readDouble()		writeDouble()	
readFully()	read a portion	writeFully()	write a portion
readLine()	read a line	writeLine()	write a line
readShort()	read short int	writeShort()	write short int
readLong()	read long int	writeLong()	write long int

Other controls in file

```
 import java.io.*;
class FileReadAll{
    public static void main(String args[]) throws
IOException{
        RandomAccessFile rafi = new
RandomAccessFile("Input.txt","r");
        RandomAccessFile rafo = new
RandomAccessFile("Output.txt","rw");
        int i = rafi.readInt();
        rafi.seek(0); // Takes the pointer back
        int i = rafi.readInt();
        rafo.writeInt(i);
        rafi.close();
        rafo.close();
    }
}
```

Assignment

1. Create a package in java to include the method of transposing a matrix.
2. Create a package in java to include a method for dealing with complex number arithmetic.
3. Write a program in java that shows itself on the output screen when executed.
4. What are the possible modes of opening a file with RandomAccessFile class?

Java and Web Technology

Multiple Inheritance and Multithreaded Programming in Java

Malay Bhattacharyya

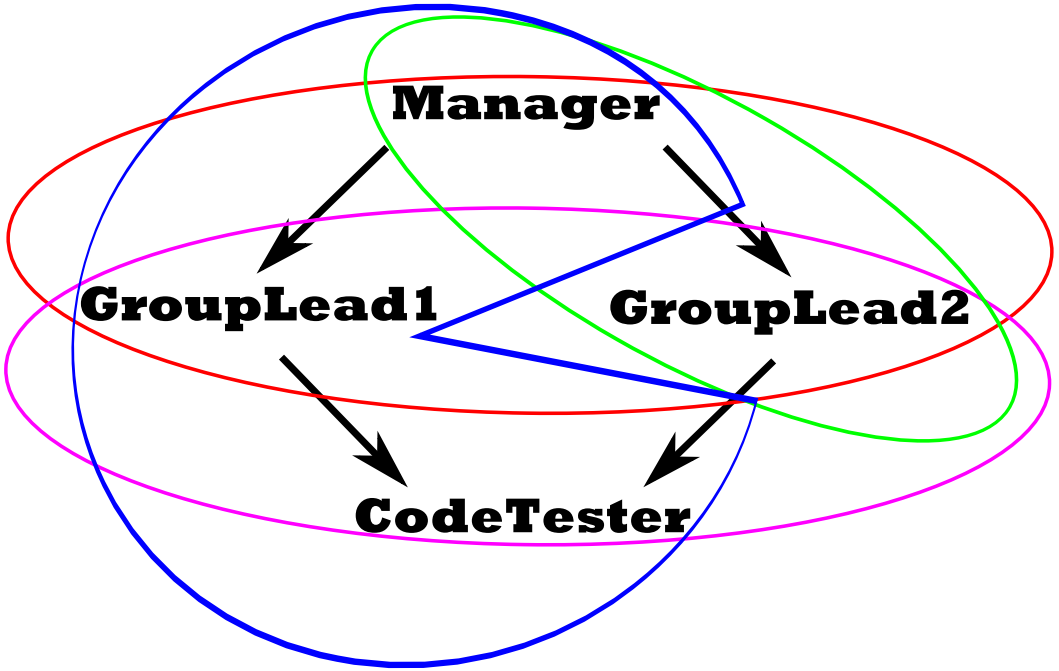
Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

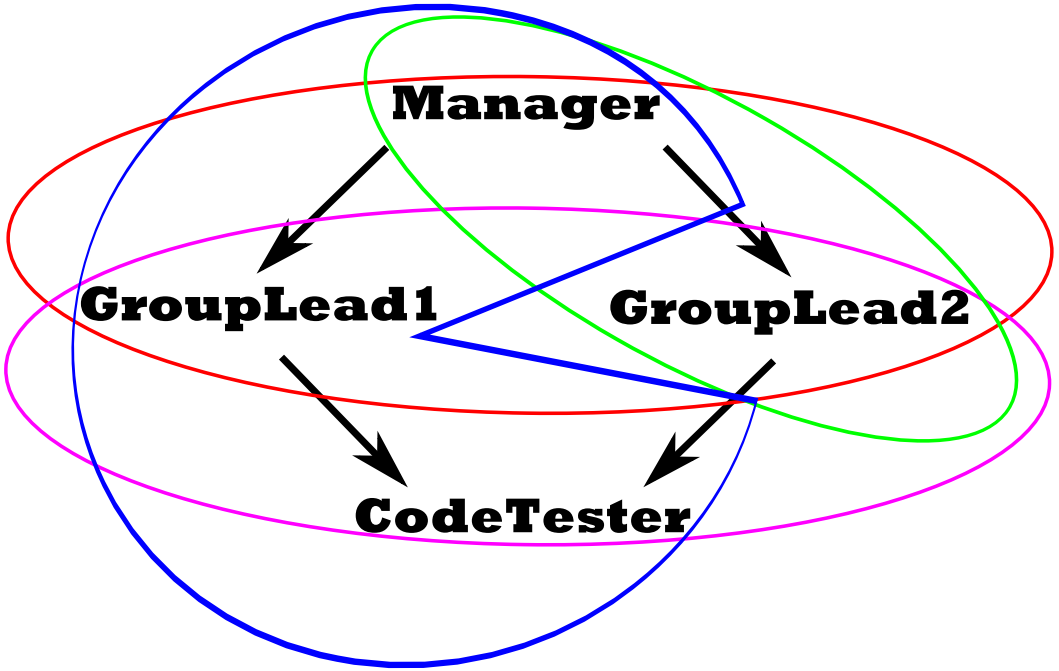
February, 2014

1. Multiple inheritance
   - Visualizing a complex form inheritance
   - Why interfaces?
   - Concept of interfaces
   - Implementing multiple inheritance
2. The concept of threads
   - What is a thread?
   - What is a multithread?
   - Life cycle of a thread
3. Use of threads in java
   - Creating a thread
   - Multithreading in java
   - Thread exceptions
4. Assignment

Visualizing a complex form inheritance



Visualizing a complex form inheritance



All forms of inheritances together – single (shown in green), multi-level (shown in blue), hierarchical (shown in red) and multiple (shown in violet).

Why interfaces?

In multiple inheritance there exists more than one superclass for a single subclass

Why interfaces?

In multiple inheritance there exists more than one superclass for a single subclass

But, this is conceptually not possible in java

Why interfaces?

In multiple inheritance there exists more than one superclass for a single subclass

But, this is conceptually not possible in java

So what?

Why interfaces?

In multiple inheritance there exists more than one superclass for a single subclass

But, this is conceptually not possible in java

So what?

We use interfaces

Concept of interfaces

An interface is nothing but a class

Concept of interfaces

An interface is nothing but a class

The characteristics of interfaces:

- Interfaces are used as *superclasses* whose properties are inherited
- Interfaces support final fields and abstract methods

Defining an interface

```
interface IF{
    // Declaration of variables (always static final)
    ...
    // Declaration of methods (always abstract)
    ...
}
```

Extending interfaces

An interface (extends) an interface

Extending interfaces

An interface (extends) an interface

```
 interface IF1{
    ...
}
interface IF2{
    ...
}
interface subIF extends IF1, IF2{ ...  }
```

Extending interfaces

An interface (extends) an interface

```
 interface IF1{
    ...
}
interface IF2{
    ...
}
interface subIF extends IF1, IF2{ ...  }
```

Note: The extension of multiple interfaces gives you the power of implementing multiple inheritance

Implementing interfaces

A class (implements) an interface

Implementing interfaces

A class (implements) an interface

```
 interface IF{
    ...
}
class subClass implements IF{ ...  }
class subClass extends superClass implements IF{ ...
}
```

Implementing interfaces

A class (implements) an interface

```
 interface IF{
    ...
}
class subClass implements IF{ ...  }
class subClass extends superClass implements IF{ ...
}
```

Note: Implementing is nothing but extending an interface to a class

Implementing multiple inheritance

The concept:

Implementing multiple inheritance

The concept:

An interface can extend multiple interfaces

Implementing multiple inheritance

The concept:

An interface can extend multiple interfaces

A class can implement that interface extending multiple interfaces

Implementing multiple inheritance – An example

```
 interface IF1{
    int i = 1;
}
interface IF2{
    int j = 2;
}
interface IF3 extends IF1, IF2{
    int k = 3;
}
class Multiple implements IF3{
    public static void main(String args[]){
        System.out.println("In main()");
        System.out.println("In IF"+i);
        System.out.println("In IF"+j);
        System.out.println("In IF"+k);
}
```

Using methods from interfaces

```
 interface IF{
    void display();
}
class Test implements IF{
    public void display(){
        System.out.println("Hi!");
    }
}
class UseIF{
    public static void main(String args[]){
        Test t = new Test();
        t.display();
    }
}
```

What is a thread?

A thread is a lightweight task

What is a thread?

A thread is a lightweight task

Main thread

$\swarrow \quad \downarrow \quad \searrow$

Thread 1 $\leftrightarrows$ Thread 2 $\leftrightarrows$ Thread 3

What is a thread?

A thread is a lightweight task

Main thread

$\swarrow \quad \downarrow \quad \searrow$

Thread 1 $\leftrightarrows$ Thread 2 $\leftrightarrows$ Thread 3

<u>Note</u>: Threads may switch or exchange data/results

What is a multithread?

A multithread is a combination of threads

What is a multithread?

A multithread is a combination of threads

Why multithreading?

What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized

What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized
- No block on input/output

What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized
- No block on input/output
- True parallel execution on multiprocessor hardware

What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized
- No block on input/output
- True parallel execution on multiprocessor hardware
- Games with intelligent user agents and animation

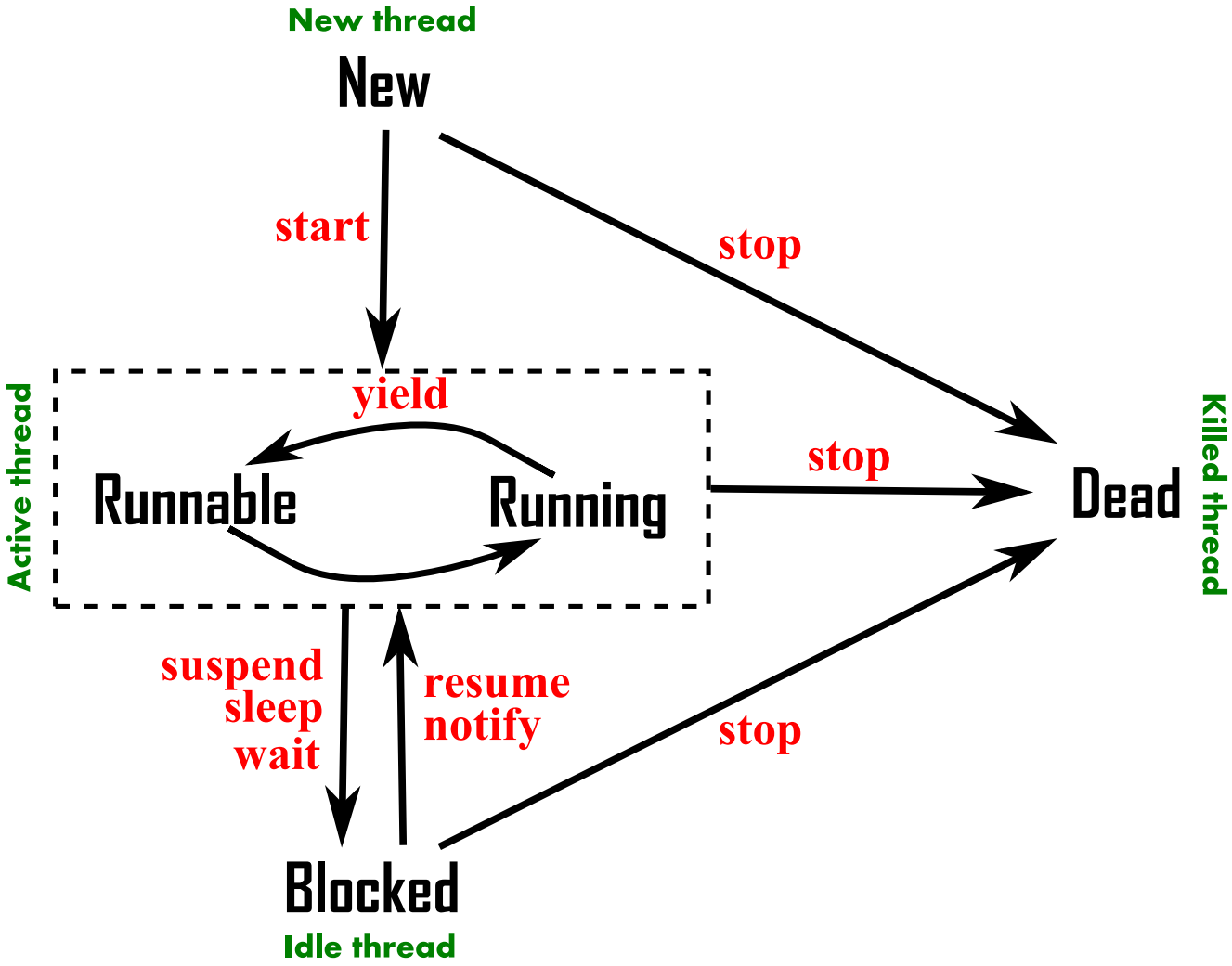
What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized
- No block on input/output
- True parallel execution on multiprocessor hardware
- Games with intelligent user agents and animation
- Automatic garbage collection hidden from the user

Life cycle of a thread



State transition diagram

Creating a thread - using the Thread class

```
 class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        mt.start(); // Start a thread
    }
}
```

Creating a thread - using the Thread class

```
 class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        mt.start(); // Start a thread
    }
}
```

Note: A thread should contain a `run()` method

Creating a thread - using the Runnable interface

```
 class MyThread implements Runnable{
    public void run(){
        System.out.println("It is runnable ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        Thread t = new Thread(mt); // Initialization
        t.start(); // Start a thread
    }
}
```

Creating a thread - using the Runnable interface

```
 class MyThread implements Runnable{
    public void run(){
        System.out.println("It is runnable ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        Thread t = new Thread(mt); // Initialization
        t.start(); // Start a thread
    }
}
```

Note: A thread is indirectly created with runnable interface

Multithreading in java

```
  class MyThread1 extends Thread{
    public void run(){
        for(int i=0;i<20;i++)
            System.out.println("Iteration "+i+" of 1 ...");
    }
}
class MyThread2 extends Thread{
    public void run(){
        for(int i=0;i<20;i++)
            System.out.println("Iteration "+i+" of 2 ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        new MyThread1().start(); // Start without declaration
        new MyThread2().start(); // Start without declaration
    }
}
```

Killing a thread

```
 class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        mt.start(); // Start a thread
        mt.stop(); // Stop a thread
    }
}
```

Blocking a thread

```
  class MyThread extends Thread{
    public void run(){
        for(i=0;i<1000;i++){
            System.out.println("It is running ...");
            if(i==100){sleep(100);} // Blocked for 100ms
            wait(); // Blocked until notified
            notify(); // Notify and start
        }
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        mt.start(); // Start a thread
        mt.suspend(); // Blocked until resumed
        mt.resume(); // Resume and start
    }
}
```

Priority of threads

```
 class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread();
        mt.setPriority(3); // Prioritization
    }
}
```

Priority of threads

```
 class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread();
        mt.setPriority(3); // Prioritization
    }
}
```

Note: The `setPriority()` method can take arguments between 1 (minimum) to 10 (maximum) to quantify priorities where a normal priority is quantified by 5 (default).

Priority of threads – Illustrating with an example

```
  class MyThread1 extends Thread{
    public void run(){
        System.out.println("Entered in MyThread1 ...");
        for(int i=1;i<=10;i++) System.out.print("[MT1:  "+i+"]");
        System.out.println("Exit from MyThread1 ...");
} }
class MyThread2 extends Thread{
    public void run(){
        System.out.println("Entered in MyThread2 ...");
        for(int i=1;i<=10;i++) System.out.print("[MT2:  "+i+"]");
        System.out.println("Exit from MyThread2 ...");
} }
class MyThread3 extends Thread{
    public void run(){
        System.out.println("Entered in MyThread3 ...");
        for(int i=1;i<=10;i++) System.out.print("[MT3:  "+i+"]");
        System.out.println("Exit from MyThread3 ...");
} }
```

Priority of threads – Illustrating with an example

```
class ThreadExample{
  public static void main(String [] args){
      MyThread1 mt1 = new MyThread1();
      MyThread2 mt2 = new MyThread2();
      MyThread3 mt3 = new MyThread3();
      mt3.setPriority(Thread.MAX_PRIORITY);
      mt2.setPriority(MyThread1.getPriority()+1);
      mt1.setPriority(Thread.MIN_PRIORITY);
      System.out.println("Start MyThread1");
      mt1.start();
      System.out.println("Start MyThread2");
      mt2.start();
      System.out.println("Start MyThread3");
      mt3.start();
      System.out.println("End of the game!!!");
  }
}
```

Thread exceptions

- **ThreadDeath:** Thread has been killed
- **InterruptedException:** It cannot be handled in the current state
- **IllegalArgumentException:** Illegal method argument
- **Exception:** Any other types of exceptions

Thread exceptions

- **ThreadDeath:** Thread has been killed
- **InterruptedException:** It cannot be handled in the current state
- **IllegalArgumentException:** Illegal method argument
- **Exception:** Any other types of exceptions

Note: These exceptions can be verified by using as arguments with try-catch.

Assignment

1. Can you visualize a new type of inheritance other than the existing ones that you think cannot be derived from single, multi-level, hierarchical and multiple inheritances?
2. Write a multithreaded program in java to read from multiple files and write on the output screen.
3. How does a thread in java change from the `Runnable` to the `Running` state?
4. Write a program in java to verify whether the threads that are created with runnable interface can support multiple inheritance.

Java and Web Technology

Applets, GUI Programming and Executables in Java

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

April, 2014

1. Applet programming in java
   - Basics
   - Implementing applets in java
   - Embedding applets into HTML
   - Running applets
   - Utilities of applets
2. The GUI in java
   - Preliminaries
   - Introducing the Swing
   - Creating GUI with Swing
   - Creating fields for texts
   - Creating buttons
   - Creating alert dialogs
3. Executable form of java
   - The concept of executables in java?
   - JAR files in java
4. Assignment

What is an applet in java?

An applet is a small application in java designed to run within another application

What is an applet in java?

An applet is a small application in java designed to run within another application

Features of java applets:

- They cannot be run directly by the operating system

What is an applet in java?

An applet is a small application in java designed to run within another application

Features of java applets:

- They cannot be run directly by the operating system
- They must execute within the JRE or within another program that includes a java plug-in

What is an applet in java?

An applet is a small application in java designed to run within another application

Features of java applets:

- They cannot be run directly by the operating system
- They must execute within the JRE or within another program that includes a java plug-in
- They are small in file size, cross-platform compatible, and highly secure

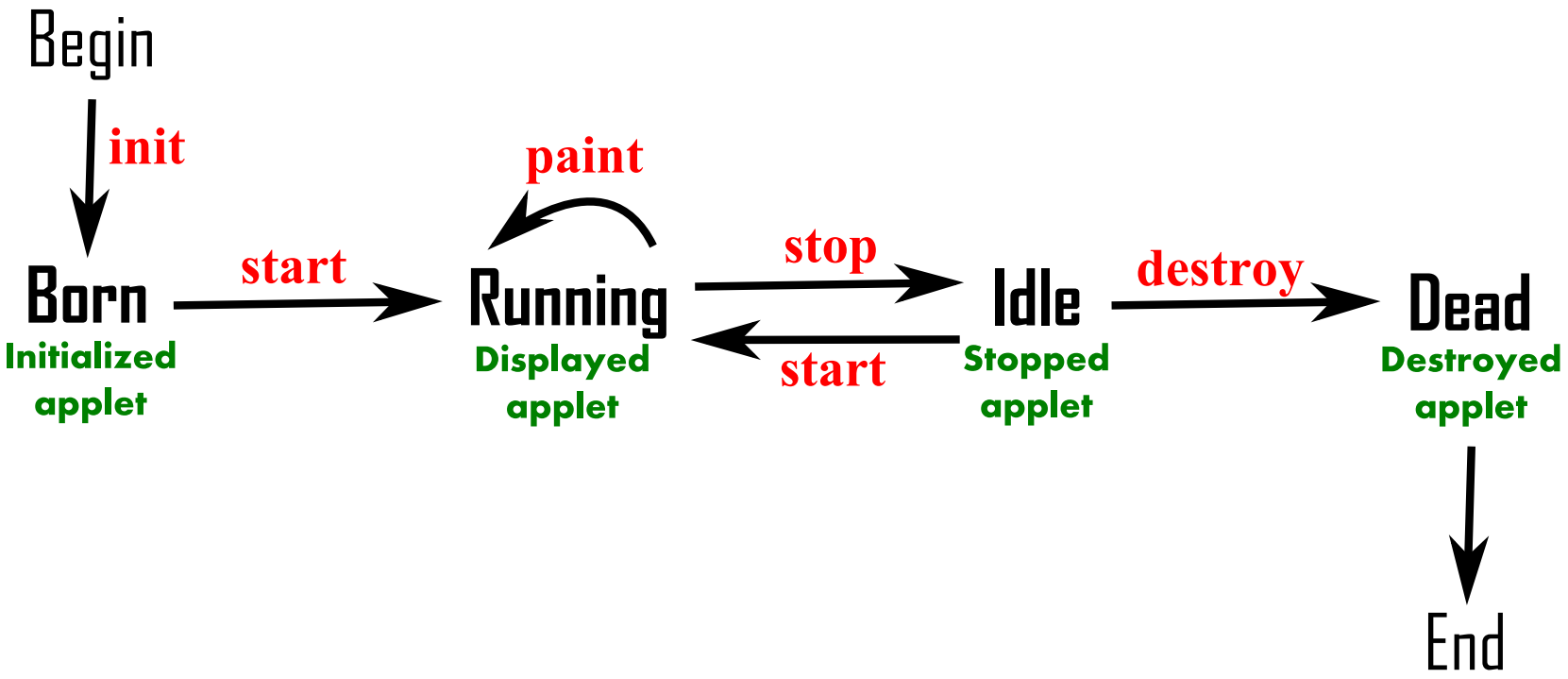
What is an applet in java?

An applet is a small application in java designed to run within another application

Features of java applets:

- They cannot be run directly by the operating system
- They must execute within the JRE or within another program that includes a java plug-in
- They are small in file size, cross-platform compatible, and highly secure
- Web browsers equipped with JVM can execute java applets

Life cycle of an applet



State transition diagram

Creating an applet

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawString("I am running", 100, 50);
    }
}
```

Creating an applet

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawString("I am running", 100, 50);
    }
}
```

Note: Almost every applet contain a `paint()` method and the class it contains takes the program name.

Other controls over an applet

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void init(){ // Initialization of an
applet
    }
    public void stop(){ // Stopping an applet
    }
    public void destroy(){ // Destroying an applet
    }
}
```

Other controls over an applet

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void init(){ // Initialization of an
applet
    }
    public void stop(){ // Stopping an applet
    }
    public void destroy(){ // Destroying an applet
    }
}
```

Note: These controls are used as per the requirements.

HTML briefs

```
 <html>
<title>
I am the title
</title>
<body>
<!  This is a comment line >
I am the red and big <font color="red"
size=18>html</font> text.
</body>
</html>
```

HTML briefs

```
 <html>
<title>
I am the title
</title>
<body>
<!  This is a comment line >
I am the red and big <font color="red"
size=18>html</font> text.
</body>
</html>
```

Note: `<html>`, `</html>`, `<title>`, etc. are html tags and they are not case sensitive.

Embedding an applet into HTML

```
 <html>
<title>
I am the title
</title>
I am running an <font color="red">applet</font>
below:
<applet code="MyApplet.class" width=400 height=300>
</applet>
</html>
```

Embedding an applet into HTML

```
 <html>
<title>
I am the title
</title>
I am running an <font color="red">applet</font>
below:
<applet code="MyApplet.class" width=400 height=300>
</applet>
</html>
```

Note: The applet class is embedded as code.

Running applets

Source applet: MyApplet.java

Running applets

Source applet: MyApplet.java

An applet program

Running applets

Source applet: MyApplet.java

An applet program

Compilation: javac MyApplet.java

Running applets

Source applet: MyApplet.java

An applet program

Compilation: javac MyApplet.java

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

Running applets

Source applet: MyApplet.java

An applet program

Compilation: javac MyApplet.java

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

Source HTML: MyHTML.html

Running applets

Source applet: MyApplet.java

An applet program

Compilation: javac MyApplet.java

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

Source HTML: MyHTML.html

An HTML page that embeds the applet class "MyApplet.class"

Running applets

Source applet: MyApplet.java

An applet program

Compilation: javac MyApplet.java

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

Source HTML: MyHTML.html

An HTML page that embeds the applet class "MyApplet.class"

Execution: appletviewer MyHTML.html

Running applets

Source applet: MyApplet.java

An applet program

Compilation: javac MyApplet.java

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

Source HTML: MyHTML.html

An HTML page that embeds the applet class "MyApplet.class"

Execution: appletviewer MyHTML.html

I am running **(in a window of size 400 by 300)**

Drawing a line

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawLine(10,20,30,40);
    }
}
```

Drawing a line

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawLine(10,20,30,40);
    }
}
```

Note: The arguments passed in `drawLine()` are position x1, position y1, position x2 and position y2, respectively.

Drawing rectangles

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        for(int i=50;i<250;i=i+5){
            g.drawRect(i+50,i,10,20);
        }
        g.drawRect(100,50,200,200); // A square
    }
}
```

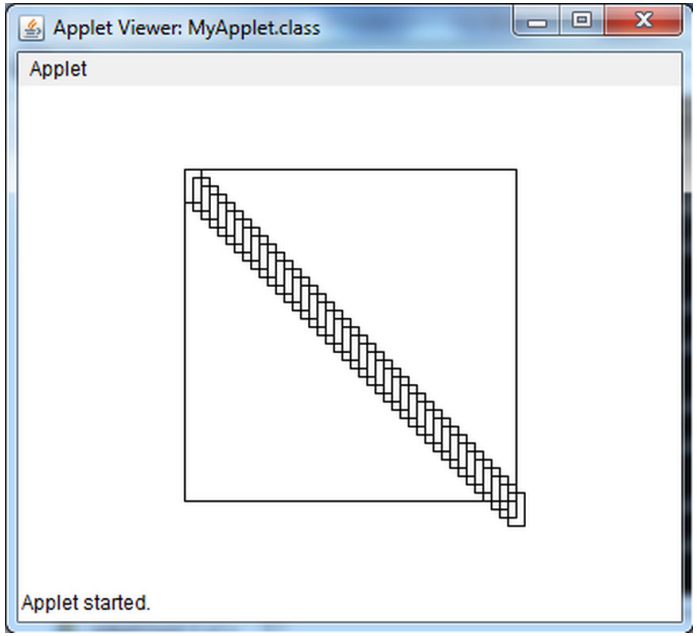
Drawing rectangles

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        for(int i=50;i<250;i=i+5){
            g.drawRect(i+50,i,10,20);
        }
        g.drawRect(100,50,200,200); // A square
    }
}
```

Note: The arguments passed in `drawRect()` are position x, position y, width and height, respectively.

Drawing rectangles

The output of the preceding program will be as follows:



Drawing arcs

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawArc(185,120,40,20,0,180);
        g.drawArc(190,130,30,30,0,360); // A circle
    }
}
```

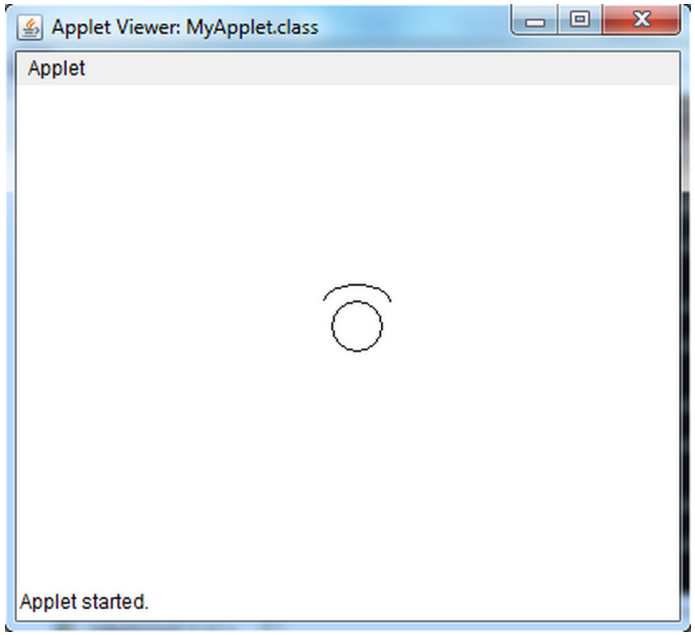
Drawing arcs

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawArc(185,120,40,20,0,180);
        g.drawArc(190,130,30,30,0,360); // A circle
    }
}
```

Note: The arguments passed in `drawArc()` are position x, position y, width, height, starting angle and arc angle, respectively.

Drawing arcs

The output of the preceding program will be as follows:



Drawing ellipses

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawOval(280,100,100,50); // Blank ellipse
        g.fillOval(200,150,50,100); // Filled ellipse
        g.drawOval(10,10,50,50); // Blank circle
        g.fillOval(120,50,100,100); // Filled circle
    }
}
```

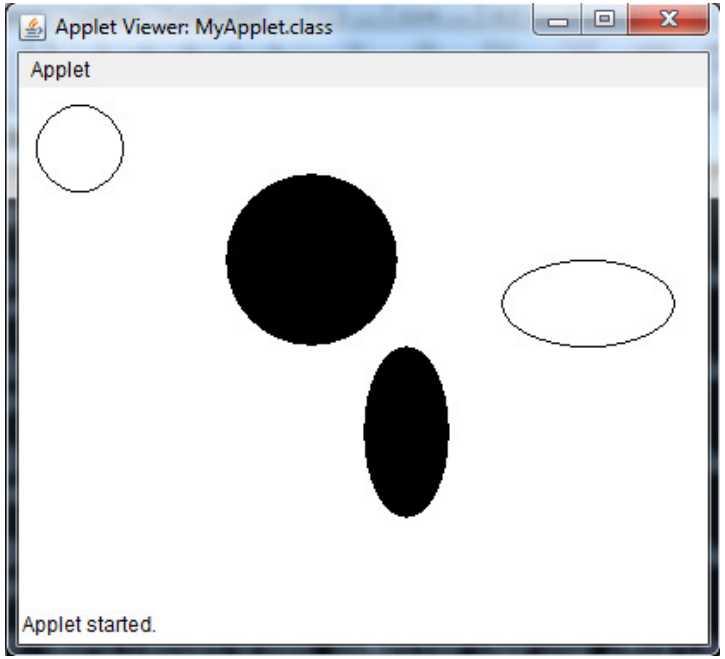
Drawing ellipses

```
 import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawOval(280,100,100,50); // Blank ellipse
        g.fillOval(200,150,50,100); // Filled ellipse
        g.drawOval(10,10,50,50); // Blank circle
        g.fillOval(120,50,100,100); // Filled circle
    }
}
```

Note: The arguments passed in `drawOval()` and `fillOval()` are position x, position y, width and height, respectively.

Drawing ellipses

The output of the preceding program will be as follows:

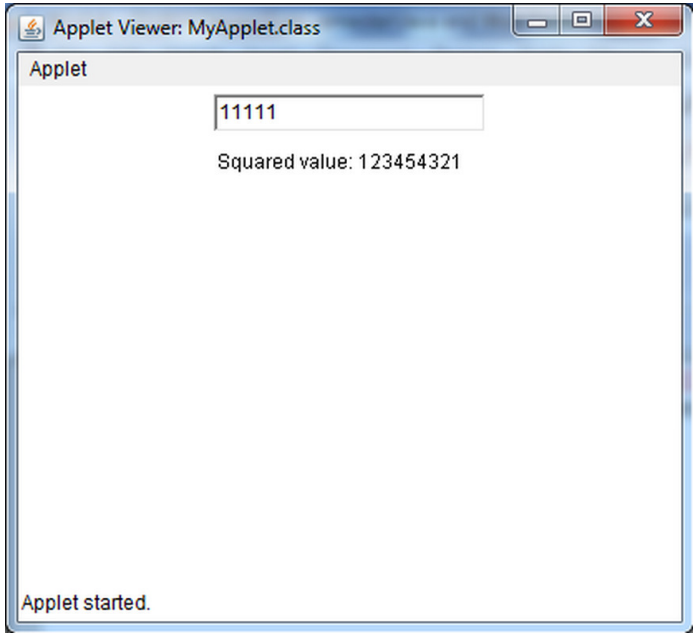


Keyboard input

```
  import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet{
    TextField TF = new TextField(20); // A field for text input
    public void init(){
        add(TF); // Add the text field to applet
    }
    public void paint(Graphics g){
        String numString, sqrString;
        numString = TF.getText(); // Retrieve text from the field
        long num = Long.parseLong(numString); // To long value
        sqrString = "Squared value:  "+String.valueOf(num * num);
        g.drawString(sqrString,120,50);
    }
    public boolean action(Event event, Object obj){
        repaint(); // Pressing enter will generate the result
        return true;
    }
}
```

Keyboard input

The output of the preceding program will be as follows:



Handling buttons

```
  import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class MyApplet extends Applet implements ActionListener{
    String message = "";
    Button BTN_YES, BTN_NO;
    public void init(){
        BTN_YES = new Button("YES");
        BTN_NO = new Button("NO");
        add(BTN_YES); BTN_YES.addActionListener(this);
        add(BTN_NO); BTN_NO.addActionListener(this);
    }
    public void actionPerformed(ActionEvent actev){
        String str = actev.getActionCommand();
        if(str.equals("YES"))     message = "Chosen YES ...";
        if(str.equals("NO"))     message = "Chosen NO ...";
        repaint();
    }
    public void paint(Graphics g){
        g.drawString(message,100,200);
    }
}
```

Other utilities

- Color
- Mouse input
- Threads and animation
- Painting
- Clocks
- 3D graphics
- etc.

Other utilities

- Color
- Mouse input
- Threads and animation
- Painting
- Clocks
- 3D graphics
- etc.

Note: The AWT (Abstract Window Toolkit) class that can be imported through `java.awt.*` is used to implement these utilities of creating graphical interface.

Preliminaries

GUI stands for Graphical User Interface

Preliminaries

GUI stands for Graphical User Interface

The platform-specific AWT class in java helps to create the GUI

Preliminaries

GUI stands for Graphical User Interface

The platform-specific AWT class in java helps to create the GUI (as we have already seen with applets)

Preliminaries

GUI stands for Graphical User Interface

The platform-specific AWT class in java helps to create the GUI (as we have already seen with applets)

But it suffers from several limitations

Preliminaries

GUI stands for Graphical User Interface

The platform-specific AWT class in java helps to create the GUI (as we have already seen with applets)

But it suffers from several limitations

– A component might appear differently on different platforms

Preliminaries

GUI stands for Graphical User Interface

The platform-specific AWT class in java helps to create the GUI (as we have already seen with applets)

But it suffers from several limitations

– A component might appear differently on different platforms
– The components cannot be easily changed due to platform dependence

Preliminaries

GUI stands for Graphical User Interface

The platform-specific AWT class in java helps to create the GUI (as we have already seen with applets)

But it suffers from several limitations

– A component might appear differently on different platforms
– The components cannot be easily changed due to platform dependence
– The components become restricted in design

Introducing the Swing

Swing is a class in java that can be imported through `javax.swing.*`

Introducing the Swing

Swing is a class in java that can be imported through `javax.swing.*`

A Swing GUI comprises two important items – *components* and *containers*

Introducing the Swing

Swing is a class in java that can be imported through `javax.swing.*`

A Swing GUI comprises two important items – *components* and *containers*

Summary of Swing components:

- Swing equivalent of AWT components (JLabel, JButton, JPanel, JSlider, etc.).
- Swing components that have no AWT equivalent (JColorChooser, JInternalFrame, JOptionPane, JToolBar, JEditorPane, etc.).
- Other simple components (JCheckBox, JRadioButton, JTextField, JTextArea, JFileChooser, etc.).

A simple example

```
  import javax.swing.*;
class MySwing{
    MySwing(){
        JFrame jf = new JFrame("Title");
        jf.setSize(275, 100);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //
Terminate the program on closing
        JLabel jl = new JLabel("Swing rocks!");
        jf.add(jl); // Add label to the container frame
        jf.setVisible(true); // Display the frame
    }
    public static void main(String args[]){
        SwingUtilities.invokeLater(new Runnable(){
            public void run(){ new MySwing(); }
        });
    }
}
```

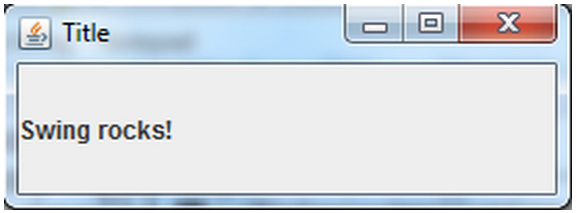
A simple example

```
  import javax.swing.*;
class MySwing{
    MySwing(){
        JFrame jf = new JFrame("Title");
        jf.setSize(275, 100);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //
Terminate the program on closing
        JLabel jl = new JLabel("Swing rocks!");
        jf.add(jl); // Add label to the container frame
        jf.setVisible(true); // Display the frame
    }
    public static void main(String args[]){
        SwingUtilities.invokeLater(new Runnable(){
            public void run(){ new MySwing(); }
        });
    }
}
```

Note: GUI is initiated with the `run()` method.

A simple example

The output of the preceding program will be as follows:



Creating fields for texts

```
  import javax.swing.*;
class MySwing{
    MySwing(){
        JFrame jf = new JFrame("Write your story:");
        jf.setSize(300, 300);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JTextField jt = new JTextField(1000);
        jf.add(jt); // Add text field to the container frame
        jf.setVisible(true); // Display the frame
    }
    public static void main(String args[]){
        SwingUtilities.invokeLater(new Runnable(){
            public void run(){ new MySwing(); } });
    }
}
```

Creating buttons

```
  import javax.swing.*;
class ButtonDemo extends JFrame{
    ButtonDemo(){
        JButton jb1 = new JButton("Button1");
        JButton jb2 = new JButton("Button2");
        jb1.setBounds(30,60,80,30);
        jb2.setBounds(150,60,80,30);
        getContentPane().setLayout(null); // To fit components
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        getContentPane().add(jb1);
        getContentPane().add(jb2);
        setSize(280,200);
        setVisible(true);
    }
    public static void main(String args[]){
        SwingUtilities.invokeLater(new Runnable(){
            public void run(){ new ButtonDemo(); } });
    }
}
```

Creating alert dialogs

```
 import javax.swing.JOptionPane;
class AlertDemo{
    public static void main(String[] args){
        String in =
JOptionPane.showInputDialog("Enter something:");
        JOptionPane.showMessageDialog(null, "You
entered " + in);
    }
}
```

The concept of executables in java?

There is no concept of *.exe files in java

The concept of executables in java?

There is no concept of *.exe files in java because java is based on bytecode

The concept of executables in java?

There is no concept of *.exe files in java because java is based on bytecode

Executables work based on JVM

Creating a JAR file

JAR stands for Java Archive

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Source code: TMain.java (with two classes)

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Source code: TMain.java (with two classes)

Compile and prepare "TMain.class" and "T.class"

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Source code: TMain.java (with two classes)

Compile and prepare "TMain.class" and "T.class"

Creation of manifest file: mnfst.txt

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Source code: TMain.java (with two classes)

Compile and prepare "TMain.class" and "T.class"

Creation of manifest file: mnfst.txt

Write "Main-Class: TMain" into it and add a new line (Must!)

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Source code: TMain.java (with two classes)

Compile and prepare "TMain.class" and "T.class"

Creation of manifest file: mnfst.txt

Write "Main-Class: TMain" into it and add a new line (Must!)

Creation of JAR: jar cvfm Out.jar mnfst.txt TMain.class T.class

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Source code: TMain.java (with two classes)

Compile and prepare "TMain.class" and "T.class"

Creation of manifest file: mnfst.txt

Write "Main-Class: TMain" into it and add a new line (Must!)

Creation of JAR: jar cvfm Out.jar mnfst.txt TMain.class T.class

Create (**c**) jar file with verbose output (**v**) to be written to a file (**f**) and include manifest file (**m**).

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Source code: TMain.java (with two classes)

Compile and prepare "TMain.class" and "T.class"

Creation of manifest file: mnfst.txt

Write "Main-Class: TMain" into it and add a new line (Must!)

Creation of JAR: jar cvfm Out.jar mnfst.txt TMain.class T.class

Create (**c**) jar file with verbose output (**v**) to be written to a file (**f**) and include manifest file (**m**).

Execution: java -jar Out.jar

Creating a JAR file

JAR stands for Java Archive

JAR files are built on the ZIP file format and have `.jar` extension

Source code: TMain.java (with two classes)

Compile and prepare "TMain.class" and "T.class"

Creation of manifest file: mnfst.txt

Write "Main-Class: TMain" into it and add a new line (Must!)

Creation of JAR: jar cvfm Out.jar mnfst.txt TMain.class T.class

Create (**c**) jar file with verbose output (**v**) to be written to a file (**f**) and include manifest file (**m**).

Execution: java -jar Out.jar

Alternatively, you can run by double clicking

Benefits of using JAR files

- **Security:** The contents of a JAR file can be digitally signed.

Benefits of using JAR files

- **Security:** The contents of a JAR file can be digitally signed.
- **Decreased download time:** An applet bundled in a JAR file can be downloaded to a browser in a single HTTP transaction without the need for opening a new connection for each file.

Benefits of using JAR files

- **Security:** The contents of a JAR file can be digitally signed.
- **Decreased download time:** An applet bundled in a JAR file can be downloaded to a browser in a single HTTP transaction without the need for opening a new connection for each file.
- **Compression:** JAR files are in compressed form for efficient storage.

Benefits of using JAR files

- **Security:** The contents of a JAR file can be digitally signed.
- **Decreased download time:** An applet bundled in a JAR file can be downloaded to a browser in a single HTTP transaction without the need for opening a new connection for each file.
- **Compression:** JAR files are in compressed form for efficient storage.
- **Packaging for extensions:** The extensions framework makes it possible to add functionality to the java core platform and the JAR file format defines its packaging.

Benefits of using JAR files

- **Security:** The contents of a JAR file can be digitally signed.
- **Decreased download time:** An applet bundled in a JAR file can be downloaded to a browser in a single HTTP transaction without the need for opening a new connection for each file.
- **Compression:** JAR files are in compressed form for efficient storage.
- **Packaging for extensions:** The extensions framework makes it possible to add functionality to the java core platform and the JAR file format defines its packaging.
- **Package Sealing:** Packages stored in JAR files can be optionally sealed to enforce version consistency.

Benefits of using JAR files

- **Security:** The contents of a JAR file can be digitally signed.
- **Decreased download time:** An applet bundled in a JAR file can be downloaded to a browser in a single HTTP transaction without the need for opening a new connection for each file.
- **Compression:** JAR files are in compressed form for efficient storage.
- **Packaging for extensions:** The extensions framework makes it possible to add functionality to the java core platform and the JAR file format defines its packaging.
- **Package Sealing:** Packages stored in JAR files can be optionally sealed to enforce version consistency.
- **Package Versioning:** A JAR file can hold data about the files it contains (e.g., vendor and version information).

Benefits of using JAR files

- **Security:** The contents of a JAR file can be digitally signed.
- **Decreased download time:** An applet bundled in a JAR file can be downloaded to a browser in a single HTTP transaction without the need for opening a new connection for each file.
- **Compression:** JAR files are in compressed form for efficient storage.
- **Packaging for extensions:** The extensions framework makes it possible to add functionality to the java core platform and the JAR file format defines its packaging.
- **Package Sealing:** Packages stored in JAR files can be optionally sealed to enforce version consistency.
- **Package Versioning:** A JAR file can hold data about the files it contains (e.g., vendor and version information).
- **Portability:** The mechanism for handling JAR files is a standard part of the java platform's core API.

Assignment

1. Is it possible to create an applet program that will contain a main() method?
2. Write an applet program in java that will show a robotic face.
3. Write a java program using Swing that can refresh a GUI on pressing a button.

Java and Web Technology

Database Connection with Java and Advanced Features

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

April, 2014

1 Introduction to JDBC
- Basics
- Connection workflow through JDBC
- Types of JDBC drivers

2 Database connection with java
- Basics about implementation
- Installing the JDBC driver
- Connecting to JDBC – Direct approach
- Connecting to JDBC – Indirect approach
- Accessing the database through JDBC

3 Advanced features of java

4 Assignment

Basics

JDBC stands for java database connectivity

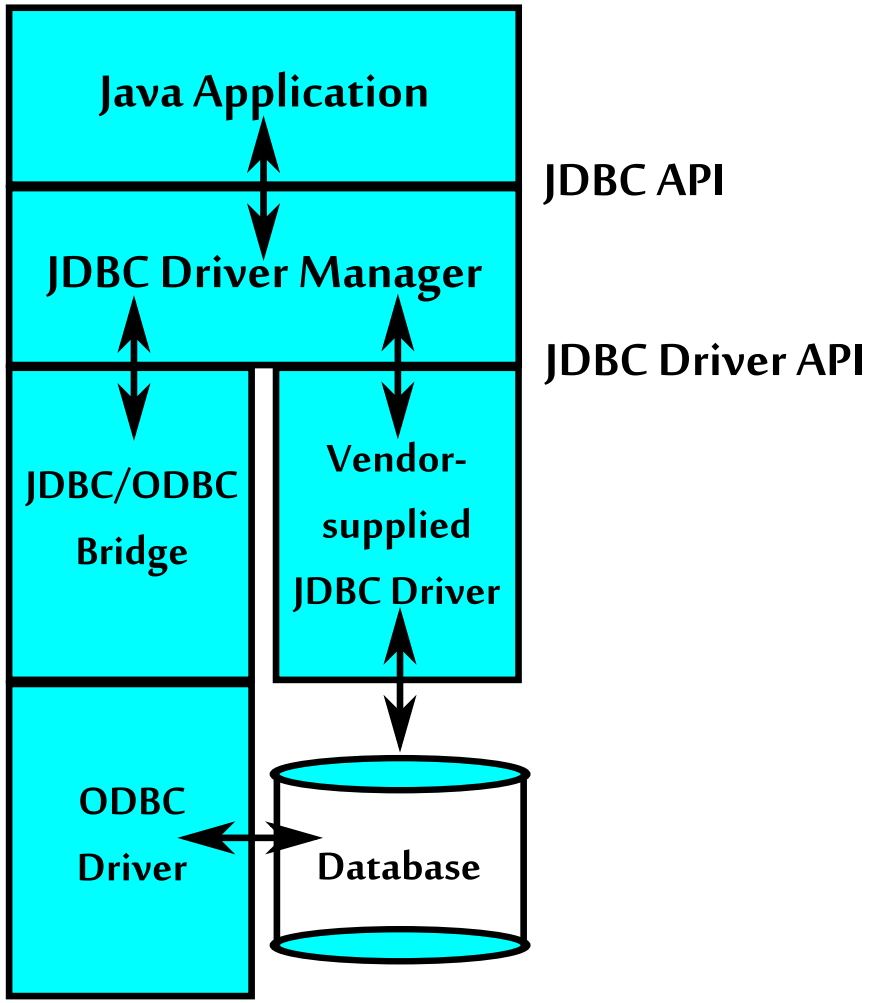
Basics

JDBC stands for java database connectivity

Features of JDBC:

- It helps to connect to a database for querying and updating
- It works based on the structured query language (SQL)
- The working principle of JDBC is platform independent
- It is compatible with any java code

Connection workflow through JDBC



Communication path between JDBC and the database

Types of JDBC drivers

- **Type 1 Driver:** translates JDBC to ODBC and relies on an ODBC driver to communicate with the database
- **Type 2 Driver:** communicates with the client API of a database
- **Type 3 Driver:** uses a database-independent protocol to communicate database requests to a server component, which then translates the requests into a database-specific protocol
- **Type 4 Driver:** translates JDBC requests directly to a database-specific protocol

Types of JDBC drivers

- **Type 1 Driver:** translates JDBC to ODBC and relies on an ODBC driver to communicate with the database
- **Type 2 Driver:** communicates with the client API of a database
- **Type 3 Driver:** uses a database-independent protocol to communicate database requests to a server component, which then translates the requests into a database-specific protocol
- **Type 4 Driver:** translates JDBC requests directly to a database-specific protocol

Note: ODBC stands for oracle database connectivity

Basics about implementation

The database connection with java, whatever be the platform, can be set up in three phases:

1. Installing the JDBC driver
2. Connecting to JDBC (this connection can be set in two different ways – direct approach and indirect approach).
3. Accessing the database through JDBC

Basics about implementation

The database connection with java, whatever be the platform, can be set up in three phases:

1. Installing the JDBC driver
2. Connecting to JDBC (this connection can be set in two different ways – direct approach and indirect approach).
3. Accessing the database through JDBC

Note: For connecting to JDBC, one has to pass the platform (driver/url) and permission (username/password) details to the JDBC. In direct approach, these details are directly feed into the program, whereas in indirect approach, they are kept in a separate file (db.properties) and linked with the program.

Installing the JDBC driver

JDBC driver can be installed and linked in two ways:

Temporary solution:

1. Download the latest and compatible JDBC driver file (e.g., '`ojdbc14.jar`' for the Oracle environment 10g).
2. Consider the path where you keep the driver file as '`fullpath`' and set the classpath of the current session by writing "`java -cp fullpath\ojdbc14.jar`"

Permanent solution:

1. Download the latest and compatible JDBC driver file (e.g., '`ojdbc14.jar`' for the Oracle environment 10g).
2. Consider the path where you keep the driver file as '`fullpath`' and set the classpath permanently by setting the value "`fullpath\ojdbc14.jar`" to the environment variable '`CLASSPATH`' (e.g., at Computer → Properties → Advanced in Windows XP/7).

Connecting to JDBC – Direct approach

```
  import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;
public class ConnectDirectOracleJDBC{
    public static void main(String args[]){
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
        }catch(ClassNotFoundException e){}
        Connection connection = null;
        try{
            connection = DriverManager.getConnection(
"jdbc:oracle:thin:@LocalHost:1521:XE","system","*****");
        }catch(SQLException e){}
        if(connection != null)
            System.out.println("Success!");
        else
            System.out.println("Failure!");
    }
}
```

Connecting to JDBC – Indirect approach

First, prepare the following file with platform and permission details.

Database properties file: db.properties

Connecting to JDBC – Indirect approach

First, prepare the following file with platform and permission details.

Database properties file: db.properties

This file contains the following information:

```
jdbc.drivers = <Full driver name>
jdbc.url = JDBC:<driver>:<option>:<Local
host/IP>:<Port number>:<Database name/SID>
jdbc.username = <user name>
jdbc.password = <password>
```

Connecting to JDBC – Indirect approach

First, prepare the following file with platform and permission details.

Database properties file: db.properties

This file contains the following information:

```
jdbc.drivers = <Full driver name>
jdbc.url = JDBC:<driver>:<option>:<Local
host/IP>:<Port number>:<Database name/SID>
jdbc.username = <user name>
jdbc.password = <password>
```

Note: This is kept along with the java program (that will access the database) in the same path.

Connecting to JDBC – Indirect approach

What exactly to write in db.properties file?

While using MySQL platform:

```
jdbc.drivers = COM.MySQL.JDBC.Driver
jdbc.url = JDBC:MySQL://LocalHost:3306/DatabaseName
jdbc.username = root
jdbc.password = *****
```

While using Oracle SQL platform:

```
jdbc.drivers = Oracle.JDBC.Driver.OracleDriver
jdbc.url = JDBC:Oracle:thin:@LocalHost:1521:XE
jdbc.username = system
jdbc.password = *****
```

Connecting to JDBC – Indirect approach

What exactly to write in db.properties file?

While using MySQL platform:

```
jdbc.drivers = COM.MySQL.JDBC.Driver
jdbc.url = JDBC:MySQL://LocalHost:3306/DatabaseName
jdbc.username = root
jdbc.password = *****
```

While using Oracle SQL platform:

```
jdbc.drivers = Oracle.JDBC.Driver.OracleDriver
jdbc.url = JDBC:Oracle:thin:@LocalHost:1521:XE
jdbc.username = system
jdbc.password = *****
```

Note: `XE` is the SID value.

Connecting to JDBC – Indirect approach

What exactly to write in db.properties file?

While using IBM DB2 platform:

jdbc.drivers = COM.ibm.db2.jdbc.app.DB2Driver
jdbc.url = JDBC:db2:COREJAVA
jdbc.username = root
jdbc.password = *****

Connecting to JDBC – Indirect approach

What exactly to write in db.properties file?

While using IBM DB2 platform:

jdbc.drivers = COM.ibm.db2.jdbc.app.DB2Driver
jdbc.url = JDBC:db2:COREJAVA
jdbc.username = root
jdbc.password = *****

While using Cloudscape platform:

jdbc.drivers = COM.cloudscape.core.JDBCDriver
jdbc.url = JDBC:cloudscape:COREJAVA;create=true
jdbc.username = root
jdbc.password = *****

Connecting to JDBC – Indirect approach

Now the connection is set as follows.

```
      public static Connection ConnectIndirectOracleJDBC()
throws SQLException, IOException {
        Properties props = new Properties();
        FileInputStream fis = new
FileInputStream("db.properties");
        props.load(fis); // Load database properties from file
        fis.close();
        String drivers = props.getProperty("jdbc.drivers");
        if(drivers != null)
            System.setProperty("jdbc.drivers", drivers);
        String url = props.getProperty("jdbc.url");
        String username = props.getProperty("jdbc.username");
        String password = props.getProperty("jdbc.password");
        return DriverManager.getConnection(url, username,
password);
    }
}
```

Accessing the database through JDBC

```
  import java.sql.*;
import java.io.*;
import java.util.*;
public class MyDatabase{
    public static void main(String args[]){
        try{
            Connection conn = ConnectIndirectOracleJDBC();
            Statement stat = conn.createStatement();
            stat.execute("CREATE TABLE TWT (Tweet CHAR(5))");
            stat.execute("INSERT INTO TWT VALUES ('Hi!')");
            ResultSet r = stat.executeQuery("SELECT * FROM TWT");
            r.next();
            System.out.println(r.getString(1));
            r.close();
            stat.close();
            conn.close(); // Close the connection
        }
        catch(SQLException ex){ }
    }
```

Advanced features of java

- **JMS:** The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API to create, send, receive, and read messages between two or more clients. It is based on the Java Enterprise Edition (Java EE) and allows the communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous.

Advanced features of java

- **JMS:** The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API to create, send, receive, and read messages between two or more clients. It is based on the Java Enterprise Edition (Java EE) and allows the communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous.
- **AJAX:** Asynchronous JavaScript and XML (AJAX) is a group of interrelated Web development techniques used on the client-side to create asynchronous Web applications. With Ajax, Web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page.

Advanced features of java

- **JMS:** The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API to create, send, receive, and read messages between two or more clients. It is based on the Java Enterprise Edition (Java EE) and allows the communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous.
- **AJAX:** Asynchronous JavaScript and XML (AJAX) is a group of interrelated Web development techniques used on the client-side to create asynchronous Web applications. With Ajax, Web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page.
- **JavaFX:** JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.

Assignment

1. Write a program in java using JDBC to remove the table `Twitter` saved in an IBM DB2 database.

Class Test 1: Java (MCA402)

1. Write a single program in java to show the use of the exception handlers try, catch, throw, throws and finally; any one type of arithmetic operator, relational operator, logical operator, bitwise operator and special operator; the packages io, util and lang; access types public and private.

15

Java and Web Technology

HTML and CSS

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

April, 2014

1 HTML
- Basics
- The first HTML code
- Comments in HTML
- Hyperlink
- Attaching figures
- Embedding videos
- Creating tables in HTML
- Other features

2 CSS
- Basics
- Linking CSS with HTML
- A simple CSS
- User-defined headers
- User-defined blocks
- User-defined bullets, numberings and hyperlinks
- Comments in CSS

3 Assignment

Basics

HTML stands for hypertext markup language

Basics

HTML stands for hypertext markup language

Tags are building blocks of an HTML program

Basics

HTML stands for hypertext markup language

Tags are building blocks of an HTML program

– Tags appear in pair (a start tag <. . .> and a matching end tag </. . .>)

– Tag names are not case sensitive

– Every tag defines a boundary within which the property applies to

– HTML tags can be nested

Basics

HTML stands for hypertext markup language

Tags are building blocks of an HTML program

– Tags appear in pair (a start tag <...> and a matching end tag </...>)
– Tag names are not case sensitive
– Every tag defines a boundary within which the property applies to
– HTML tags can be nested

E.g., `<html>`, `</html>`, `<title>`, `</title>`, etc.

The first HTML code – An example

```
 <html>
<title>
I am the title
</title>
<body>
I am the red and big <font color="red"
size=18>html</font> text.
</body>
</html>
```

The first HTML code – Title, body, paragraph, space and newline

`<title>...</title>`: This defines the title

The first HTML code – Title, body, paragraph, space and newline

`<title>...</title>`: This defines the title

`<body>...</body>`: This defines the body of the html

The first HTML code – Title, body, paragraph, space and newline

`<title>...</title>`: This defines the title

`<body>...</body>`: This defines the body of the html

`<p>...</p>`: This defines a paragraph

The first HTML code – Title, body, paragraph, space and newline

`<title>...</title>`: This defines the title

`<body>...</body>`: This defines the body of the html

`<p>...</p>`: This defines a paragraph

` `: This defines a space

The first HTML code – Title, body, paragraph, space and newline

`<title>...</title>`: This defines the title

`<body>...</body>`: This defines the body of the html

`<p>...</p>`: This defines a paragraph

` `: This defines a space

`<br/>`: This defines a new line

Comments in HTML

```
 <html>
<!  This is a comment line >
<title>
I am the title
</title>
</html>
```

Comments in HTML

```
 <html>
<!  This is a comment line >
<title>
I am the title
</title>
</html>
```

Note: Comments cannot be nested

Hyperlink

```
<a href = "http://isical.ac.in/~malay_r">Click here</a>
```

Attaching figures

```
<img src = "Photo.jpg" align="center" height=180
width=240></img>
```

Attaching figures

```
<img src = "Photo.jpg" align="center" height=180
width=240></img>
```

Note: The files `*.png` and `*.bmp` can also be included like this

Embedding videos

```
<video height=200 width=250 controls>
   <source src="Movie.mp4" type="video/mp4">
</video>
```

Embedding videos

```
<video height=200 width=250 controls>
   <source src="Movie.mp4" type="video/mp4">
</video>
```

Note: The files `*.ogg` and `*.webm` can also be included like this

Embedding videos

```
<video height=200 width=250 controls>
   <source src="Movie.mp4" type="video/mp4">
</video>
```

Note: The files `*.ogg` and `*.webm` can also be included like this

Alternatively, you can use the following:

```
<object data="Movie.mp4" width="320" height="240">
<embed src="Movie.swf" width="320" height="240">
</object>
```

Creating tables in HTML

```
 <table border="1" align="center" bgcolor="red"
width="60" height="40">
<tr>
<td>Row 1, Column 1</td>
<td>Row 1, Column 2</td>
<td>Row 1, Column 3</td>
</tr>
<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
<td>Row 2, Column 3</td>
</tr>
</table>
```

Bullets and numbering

Bullets (Unordered lists):

```
<ul>
    <li> Item I </li>
    <li> Item II </li>
    <li> Item III </li>
</ul>
```

Bullets and numbering

Bullets (Unordered lists):

```
<ul>
    <li> Item I </li>
    <li> Item II </li>
    <li> Item III </li>
</ul>
```

Numberings (Ordered lists):

```
<ol>
    <li> Point 1 </li>
    <li> Point 2 </li>
    <li> Point 3 </li>
</ol>
```

Font properties

Defining font colors:

```
<font color=#FF0000> ...  </font>
```

Font properties

Defining font colors:

```
<font color=#FF0000> ...  </font>
```

Defining font types:

```
<font type="Arial"> ...  </font>
```

Font properties

Defining font colors:

```
<font color=#FF0000> ...  </font>
```

Defining font types:

```
<font type="Arial"> ...  </font>
```

Defining font sizes:

```
<font size=14> ...  </font>
```

Font properties

Defining font colors:

```
<font color=#FF0000> ...  </font>
```

Defining font types:

```
<font type="Arial"> ...  </font>
```

Defining font sizes:

```
<font size=14> ...  </font>
```

Note: Colors are defined as #RRGGBB where RR, GG and BB denote a color value between 00-FF constituting the final color

Defining text properties

Bold text:

```
<b> ...  </b>
```

Defining text properties

Bold text:

```
<b> ...  </b>
```

Italicized text:

```
<i> ...  </i>
```

Defining text properties

Bold text:

```
<b> ...  </b>
```

Italicized text:

```
<i> ...  </i>
```

Underlined text:

```
<u> ...  </u>
```

Defining text properties

Bold text:

```
<b> ... </b>
```

Italicized text:

```
<i> ... </i>
```

Underlined text:

```
<u> ... </u>
```

Subscripted and superscripted text:

```
<sub> ... </sub>
<sup> ... </sup>
```

Defining text properties

Bold text:

```
<b> ... </b>
```

Italicized text:

```
<i> ... </i>
```

Underlined text:

```
<u> ... </u>
```

Subscripted and superscripted text:

```
<sub> ... </sub>
<sup> ... </sup>
```

So, n_k^2 can be represented with the following HTML code.

```
<i>n<sub>k</sub><sup>2</sup></i>
```

Text animation

Moving text:

```
<marquee> ...  </marquee>
```

Text animation

Moving text:

```
<marquee> ...  </marquee>
```

It can have the following direction (of movement) parameters:

- `<marquee direction="left"> ...  </marquee>` **(default)**
- `<marquee direction="right"> ...  </marquee>`
- `<marquee direction="down"> ...  </marquee>`
- `<marquee direction="top"> ...  </marquee>`

Text animation

Moving text:

```
<marquee> ...  </marquee>
```

It can have the following direction (of movement) parameters:

- `<marquee direction="left"> ...  </marquee>` **(default)**
- `<marquee direction="right"> ...  </marquee>`
- `<marquee direction="down"> ...  </marquee>`
- `<marquee direction="top"> ...  </marquee>`

The other important parameters are behavior (e.g., `behavior="scroll"`) and delay of scrolling in milliseconds (e.g., `scrolldelay="200"`).

Basics

CSS stands for cascading style sheet

Basics

CSS stands for cascading style sheet

Some facts about CSS:

Basics

CSS stands for cascading style sheet

Some facts about CSS:

– HTML files linked to a CSS conforms to the properties defined in it

– Style properties defined in a CSS file overwrite the corresponding default browser values

– If a property is not specified in CSS, it will be inherited from the parent element

– The properties in a CSS file are ordered as per their ascending priorities

Linking CSS with HTML

```
<link href="style.css" rel="stylesheet"
type="text/css" media="screen" />
```

Note: This can appear anywhere in the code (certainly within `<html>` and `</html>`).

A simple CSS

```
body{
    margin: 0;
    background: #FFFFFF;
    line-height: 25px;
    text-align: justify;
    font-family: Tahoma, Arial, Helvetica,
sans-serif;
    font-size: 11px;
    color: #000000;
}
```

User-defined headers – Defining the style

```
 h1{
    font-size:  2em;
}
h2{
    font-size:  1.4em;
}
h3{
    font-size:  1em;
}
```

User-defined headers – Using in HTML

```
 <html>
<h1>I am Big header</h1>
<h2>I am Medium header</h2>
<h3>I am Small header</h3>
</html>
```

User-defined blocks – Defining the style

```
#left{
    position: fixed;
    top: 0;
    left: 50px;
    width: 150px;
    height: 100%;
    border-left: 50px solid #AA0000;
    padding: 10px;
}
```

User-defined blocks – Using in HTML

```
 <div id="left">
I am fine here :)
</div>
```

All the properties set for the block ‘left’ will be applicable here.

User-defined bullets, numberings and hyperlinks

```
 ul{
margin-left:  2em;
}
```

User-defined bullets, numberings and hyperlinks

```
 ul{
margin-left:  2em;
}

 ol{
margin-left:  3em;
}
```

User-defined bullets, numberings and hyperlinks

```
 ul{
margin-left:  2em;
}

 ol{
margin-left:  3em;
}

 a{
color:  #FFFFFF;
}
```

Comments in CSS

Single-line/multi-line comment:

/* ... ... */

/* ...

...

... */

Comments in CSS

Single-line/multi-line comment:

/* ... ... */

/* ...

...

... */

Note: Comments cannot be nested

Assignment

1. Use HTML and CSS to prepare your academic webpage comprising the latest photo and linked to your current resume.
2. Write an HTML code to show radio button in a page.

Java and Web Technology

JavaScript

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

April, 2014

1 Basics
- What is JavaScript?
- Client-server interaction
- Server side programming
- Client side programming

2 Introduction to JavaScript
- Features of JavaScript
- Writing the first JavaScript program

3 Programming in JavaScript

4 JavaScript Objects
- Concept of objects
- The objects in JavaScript

5 Assignment

What is JavaScript?

"Java is to JavaScript what Car is to Carpet" – Chris Heilmann.

What is JavaScript?

"Java is to JavaScript what Car is to Carpet" – Chris Heilmann.

JavaScript was developed at Netscape by Brendan Eich in 1995

What is JavaScript?

"Java is to JavaScript what Car is to Carpet" – Chris Heilmann.

JavaScript was developed at Netscape by Brendan Eich in 1995

May, 1995 ... **December, 1995**

What is JavaScript?

"Java is to JavaScript what Car is to Carpet" – Chris Heilmann.

JavaScript was developed at Netscape by Brendan Eich in 1995

May, 1995	...	**December, 1995**
Mocha →	LiveScript →	JavaScript

What is JavaScript?

"Java is to JavaScript what Car is to Carpet" – Chris Heilmann.

JavaScript was developed at Netscape by Brendan Eich in 1995

May, 1995	...	**December, 1995**
Mocha →	LiveScript →	JavaScript

JavaScript is a client-side scripting language

What is JavaScript?

"Java is to JavaScript what Car is to Carpet" – Chris Heilmann.

JavaScript was developed at Netscape by Brendan Eich in 1995

May, 1995	...	**December, 1995**
Mocha →	LiveScript →	JavaScript

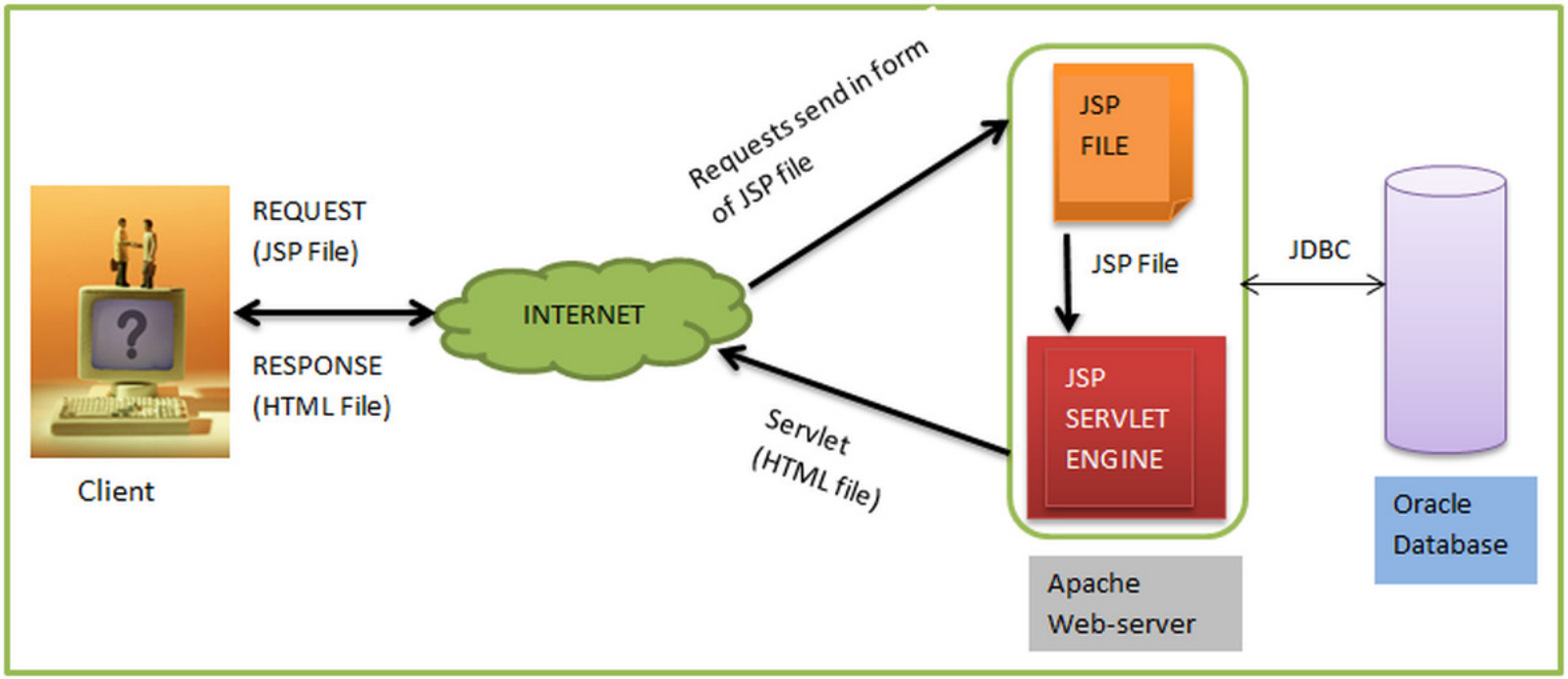
JavaScript is a client-side scripting language

Note: Scripting language is a high-level programming language that is interpreted by another program at runtime.

Client-server interaction

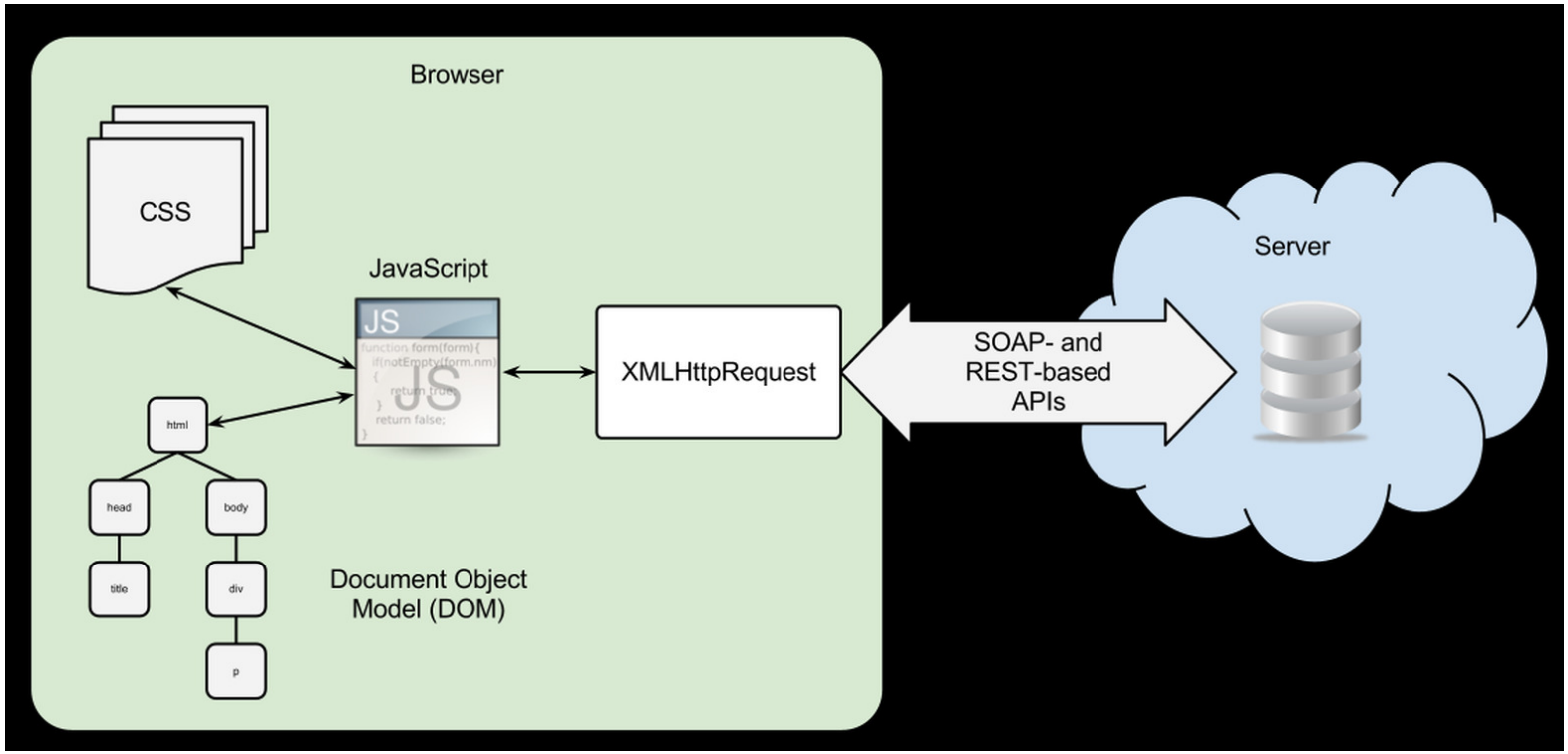
Interaction between a server and different clients

Server side programming



A case of managing server side programming with JSP

Client-side programming



A case of managing client side programming with JavaScript

Features of JavaScript

- It is a scripting language.

Features of JavaScript

- It is a scripting language.
- It is an interpreted language.

Features of JavaScript

- It is a scripting language.
- It is an interpreted language.
- It is a functional programming language.

Features of JavaScript

- It is a scripting language.
- It is an interpreted language.
- It is a functional programming language.
- It supports object-based programming.

Features of JavaScript

- It is a scripting language.
- It is an interpreted language.
- It is a functional programming language.
- It supports object-based programming.
- It is widely used by the browsers and supported by html.

Features of JavaScript

- It is a scripting language.
- It is an interpreted language.
- It is a functional programming language.
- It supports object-based programming.
- It is widely used by the browsers and supported by html.
- It can be used on client side as well as on server side.

Some facts about JavaScript

- JavaScript code is case sensitive.

Some facts about JavaScript

- JavaScript code is case sensitive.
- JavaScript is a loosely typed language.

Some facts about JavaScript

- JavaScript code is case sensitive.
- JavaScript is a loosely typed language.
- JavaScript is a freeform language.
  – White spaces between the words and tabs are ignored in any JavaScript code.
  – Line breaks are ignored except within a statement.

Some facts about JavaScript

- JavaScript code is case sensitive.
- JavaScript is a loosely typed language.
- JavaScript is a freeform language.
  – White spaces between the words and tabs are ignored in any JavaScript code.
  – Line breaks are ignored except within a statement.
- JavaScript statements end with a semicolon.

Writing the first JavaScript program

JavaScript programs can be of two types – Internal and External

Writing the first JavaScript program

JavaScript programs can be of two types – Internal and External

<u>Source</u>: Welcome2JavaScript.html

Writing the first JavaScript program

JavaScript programs can be of two types – Internal and External

Source: Welcome2JavaScript.html

```
 <html>
<script type = "text/javascript">
document.write("Welcome to JavaScript"); // Internal
</script>
</html>
```

Writing the first JavaScript program

JavaScript programs can be of two types – Internal and External

Source: Welcome2JavaScript.html

```
 <html>
<script type = "text/javascript">
document.write("Welcome to JavaScript"); // Internal
</script>
</html>
```

Source: Welcome2JavaScript.html and JavaScript.js

Writing the first JavaScript program

JavaScript programs can be of two types – Internal and External

Source: Welcome2JavaScript.html

```
 <html>
<script type = "text/javascript">
document.write("Welcome to JavaScript"); // Internal
</script>
</html>
```

Source: Welcome2JavaScript.html and JavaScript.js

```
 <html>
<script language="JavaScript" src="JavaScript.js"
type="text/javascript"/> // External
</script>
</html>
```

Data types and variables

JavaScript supports dynamic data types, i.e., the same variable can be used as different types.

Data types and variables

JavaScript supports dynamic data types, i.e., the same variable can be used as different types.

```
 var pi; // pi is an undefined variable
pi = 3.14 // variable pi is assigned a real constant
var pi = 314e-2; // pi is declared and assigned
var a = true; // a is assigned a boolean value
var a = 6; // a is assigned a number
var a = "It's MCA"; // a is assigned a string
var a = "Say 'MCA'"; // a is assigned a string
var a = 'Say again "MCA"'; // a is assigned a string
```

Arrays

```
 var myArray = new Array(); // Declaration
myArray[0] = "Om"; // Initalization
myArray[1] = "Jai"; // Initalization
myArray[2] = "Jagdish"; // Initalization
oddNumArray = new Array(1,3,5,7,9,11,13,15,17,19); //
Initialization while declaring
minArray = new Array();
minArray["favcolor"] = "cyan";
minArray["favfood"] = "Rasgulla";
```

Operators

Arithmetic operators: `+`, `-`, `*`, `/`, `%`
Relational operators: `<`, `<=`, `>`, `>=`, `==`, `!=`, `===` (equality without type conversion)
Logical operators: `&&`, `||`, `!`
Assignment operators: `=`
Increment and decrement operators: `++`, `--`
Special operators: `.`

Operators

Arithmetic operators: `+`, `-`, `*`, `/`, `%`
Relational operators: `<`, `<=`, `>`, `>=`, `==`, `!=`, `===` (equality without type conversion)
Logical operators: `&&`, `||`, `!`
Assignment operators: `=`
Increment and decrement operators: `++`, `--`
Special operators: `.`

Note: The '+' operator can be used to concatenate two strings or even a string with another data type.

Functions

An example with html

```
 <html>
<body>
<script language="javascript" type="text/javascript">
function pass(language){
    return "Welcome to " + language;
}
var JS = "JavaScript";
document.body.innerHTML = pass(JS);
</script>
</body>
</html>
```

Methods

Methods are pre-defined functions for specific usage

Methods

Methods are pre-defined functions for specific usage

Getting elements by index:

```
 alert("Hello world!");
bodyObject = document.getElementById("bodyDiv");
lowercaseString = myString.toLowerCase();
```

Methods

Methods are pre-defined functions for specific usage

Getting elements by index:

```
 alert("Hello world!");
bodyObject = document.getElementById("bodyDiv");
lowercaseString = myString.toLowerCase();
```

Accessing characters of a string:

```
 var name = "Bible";
var nameBegin = name.charAt(0);
```

Branching

Single branching:

```
if(name == "Kunal") alert("Hi Kunal!");
```

Branching

Single branching:

```
if(name == "Kunal") alert("Hi Kunal!");
```

Nested branching:

```
if(name == "Anindita"){
    alert("Anindita wins!");
}
else{
    if(name == "Shree Priya"){
        alert("Shree Priya wins!");
    }
    else{
        alert("It's a tie!");
    }
}
```

Looping

For loop:

```
 var namesArray = new
Array("Subrata","Rajasree","Haider");
for(i=0;i<namesArray.length;i++){
    alert("Hello, " + namesArray[i] + "!");
}
```

Looping

For loop:

```
 var namesArray = new
Array("Subrata","Rajasree","Haider");
for(i=0;i<namesArray.length;i++){
    alert("Hello, " + namesArray[i] + "!");
}
```

While loop:

```
 i = 0;
while(i<namesArray.length){
    alert("Hello, " + namesArray[i] + "!");
    i++;
}
```

Comments

Single-line comment:

// ...

Comments

Single-line comment:

// ...

Multi-line comment:

/* ...

...

... */

Comments

Single-line comment:

// ...

Multi-line comment:

/* ...

...

... */

HTML-style comment:

<!- - ... - ->

Comments

Single-line comment:

// ...

Multi-line comment:

/* ...
...
... */

HTML-style comment:

<!- - ... - ->

Note: Comments cannot be nested

Event handling

If the body has finished loading, show the alert message.

```
<body onLoad = alert("Loaded ...")>
```

Event handling

If the body has finished loading, show the alert message.

```
<body onLoad = alert("Loaded ...")>
```

If double clicked on the body show a user-defined message.

```
<html>
<script type = "text/javascript">
function myMessage(){
alert("Double clicked ...");
} </script>
<body onDblClick = "myMessage()">
</body> </html>
```

Event handling

If the user has clicked on the link, open the `popup.html` in a separate window.

```
<a href="#" onClick = "window.open(‘popup.html’, ‘myPopUp’)">
```

Event handling

If the user has clicked on the link, open the `popup.html` in a separate window.

```
<a href="#" onClick = "window.open(‘popup.html’, ‘myPopUp’)">
```

If the mouse has moved over the link, swap the image `cat_active` with `cat_inactive` when the mouse moves out.

```
<a href="#" onMouseover = "swapImage(‘cat_active’)" onMouseout = "swapImage(‘cat_inactive’)">
```

Event handling

Event	Description
onClick	User clicked on this item
onDblClick	User double-clicked on this item
onError	An error occurred while loading an image
onBlur	A form field lost the focus (User moved to another field)
onChange	The contents of a field has changed
onFocus	User just moved into this form element
onKeyDown	A key was pressed
onKeyUp	A key was released
onLoad	This object (iframe/image/script) finished loading
onMouseDown	A mouse button was pressed
onMouseMove	The mouse moved
onMouseOut	A mouse moved off of this element
onMouseOver	The mouse moved over this element
onMouseUp	The mouse button was released
onReset	A form's `Reset` button was pressed
onResize	The window or frame was resized
onSelect	Text has been selected
onSubmit	A form's `Submit` button has been pressed
onUnload	The user is navigating away from a page

Concept of objects

The objects in JavaScript represent collections of values

Concept of objects

The objects in JavaScript represent collections of values
– Objects with ordered collection of values: Arrays
– Objects with unordered collections of values: Objects with general properties that can be accessed as `object.property`

Concept of objects

The objects in JavaScript represent collections of values
– Objects with ordered collection of values: Arrays
– Objects with unordered collections of values: Objects with general properties that can be accessed as `object.property`

A Web page is made up of dozens of objects called the Document Object Model (DOM). Each element of the DOM has predefined properties with values derived from how that particular page was coded.

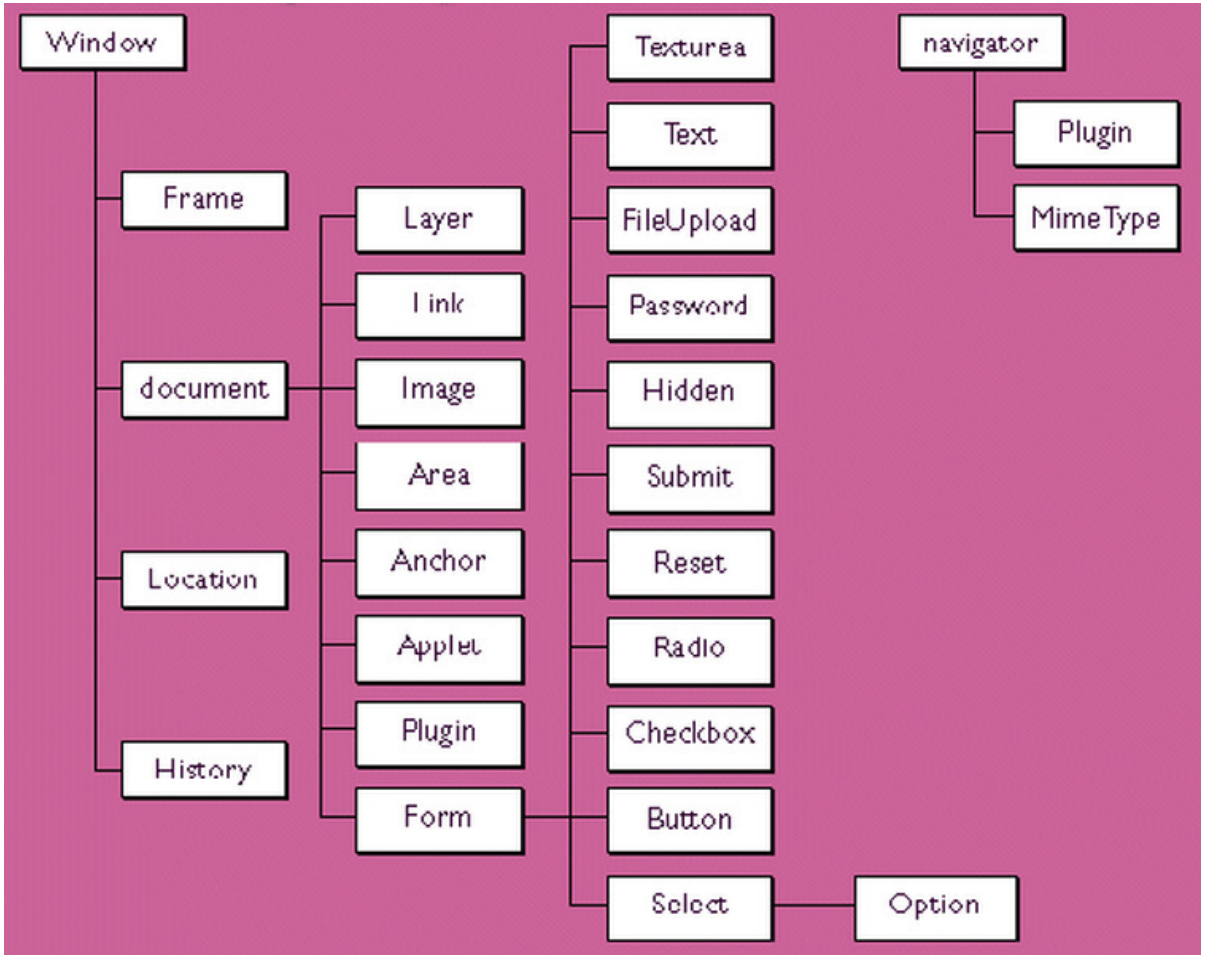
Concept of objects

The objects in JavaScript represent collections of values
– Objects with ordered collection of values: Arrays
– Objects with unordered collections of values: Objects with general properties that can be accessed as `object.property`

A Web page is made up of dozens of objects called the Document Object Model (DOM). Each element of the DOM has predefined properties with values derived from how that particular page was coded.

```
 var workshop = new Object();
workshop.title = "Introduction to JavaScript";
alert("Welcome to " + workshop.title);
```

The objects in JavaScript



The hierarchy of objects in JavaScript

Assignment

1. Write a JavaScript code that will take an identifer as user input from the client side and will retrieve and show the corresponding details against the identifier.
2. Implement an event handler using JavaScript to open the GMAIL main page on clicking a hyperlink.
3. Write a single JavaScript code to show the utility of all the event handlers.

Java and Web Technology

JSP and Servlet

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

May, 2014

1 Basics
- What is JSP?
- What is Servlet?
- Relating JSP with Servlet

2 JSP programming
- Basics
- Programming in JSP
- Objects in JSP

3 Servlet programming
- Basics
- Programming in Servlet

4 Assignment

What is JSP?

JSP stands for "java server pages"

What is JSP?

JSP stands for "java server pages"

JSP is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types.

What is JSP?

JSP stands for "java server pages"

JSP is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types.

1998: A specification of Sun Microsystems first appeared

What is JSP?

JSP stands for "java server pages"

JSP is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types.

1998: A specification of Sun Microsystems first appeared
1999: The official versions, 1.0 and 1.1, both published by Sun Microsystems

What is Servlet?

Servlet is a class, belonging to the java programming language, which is used to extend the capabilities of a server.

What is Servlet?

Servlet is a class, belonging to the java programming language, which is used to extend the capabilities of a server.

1996: The Servlet specification is created by Sun Microsystems

What is Servlet?

Servlet is a class, belonging to the java programming language, which is used to extend the capabilities of a server.

1996: The Servlet specification is created by Sun Microsystems
1997: The Servlet version 1.0 is finalized by Sun Microsystems

Connecting JSP with Servlet

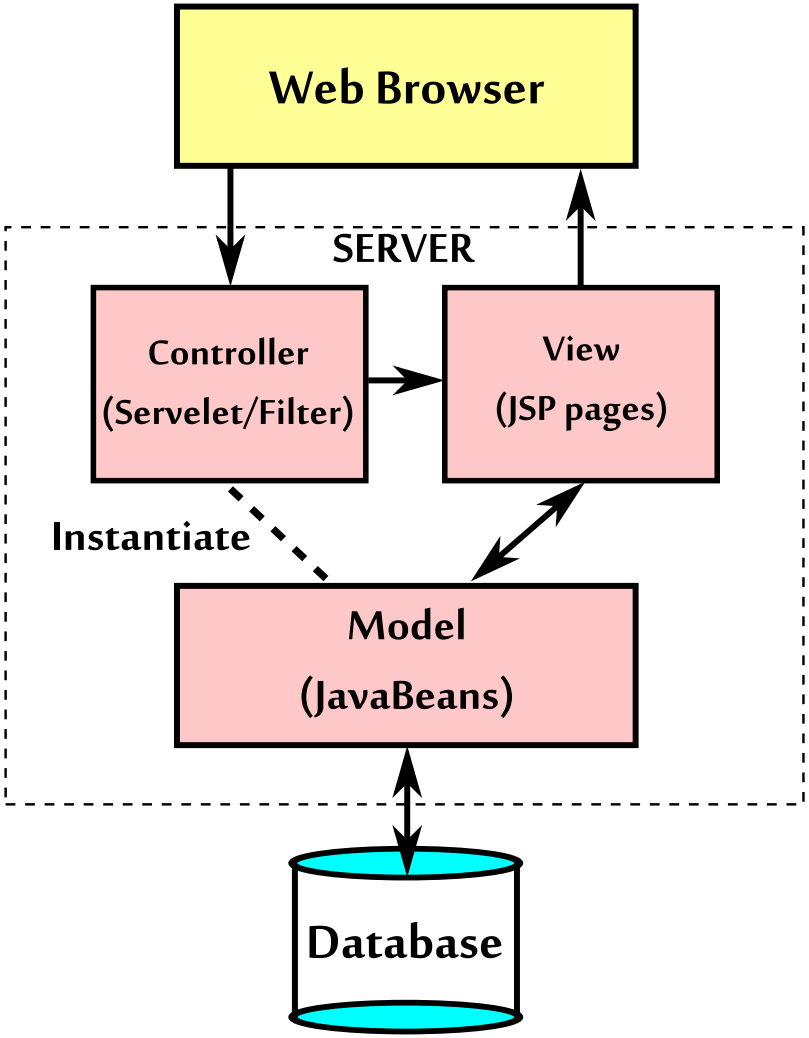
J2EE (Java 2 Enterprise Edition) is a specification for developing enterprise and distributed applications from JavaSoft (Sun Microsystems).

Connecting JSP with Servlet

J2EE (Java 2 Enterprise Edition) is a specification for developing enterprise and distributed applications from JavaSoft (Sun Microsystems).

J2EE encompasses a large set of technologies – JSP, Servlets, Enterprise JavaBeans (EJB), JDBC, Java Naming and Directory Interface (JNDI), Support for XML, Java Messaging, Java Transaction Support, JavaMail, Java support for CORBA, etc.

Connecting JSP with Servlet



The architecture of JSP Model 2

Differentiating JSP from Servlet

"JSP is java in html, Servlet is html in java" – Anonymous.

Differentiating JSP from Servlet

"JSP is java in html, Servlet is html in java" – Anonymous.

- JSP is a webpage scripting language that can generate dynamic content while Servlets are Java programs that are already compiled which also creates dynamic web content
- Servlets run faster compared to JSP
- JSP can be compiled into Java Servlets
- It's easier to code in JSP than in Java Servlets
- In MVC, JSP act as a view and Servlet act as a controller
- JSP is preferred (generally) for limited processing of data, but Servlets are best for more processing and manipulation.
- In JSP programming we can build custom tags which can directly call Java beans, but it is not possible with Servlets.
- We can achieve functionality of JSP at client side by running JavaScript at client side. There are no such methods for Servlets.

Using JSP and Servlets

Some web servers or application servers that are capable of running JSP and Servlets – Blazix from Desiderata Software, Tomcat from Apache, WebLogic from BEA Systems, WebSphere from IBM, etc.

An example of web server set up: Tomcat

Apache Tomcat is an open source software implementation of the JSP and Servlet technologies.

An example of web server set up: Tomcat

Apache Tomcat is an open source software implementation of the JSP and Servlet technologies.

To set up Tomcat, do the following:

1. Download the latest and compatible version of Apache Tomcat (from `http://tomcat.apache.org`).
2. Unpack the binary distribution into a convenient location (e.g., in `C:\apache-tomcat-5` for Windows XP/7, `/usr/local/apache-tomcat-5` for Linux/UNIX).
3. Setting `CATALINA_HOME` and `CLASSPATH` environment variables pointing to these locations.
4. Start Apache Tomcat.

An example of web server set up: Tomcat

Apache Tomcat is an open source software implementation of the JSP and Servlet technologies.

To set up Tomcat, do the following:

1. Download the latest and compatible version of Apache Tomcat (from `http://tomcat.apache.org`).
2. Unpack the binary distribution into a convenient location (e.g., in `C:\apache-tomcat-5` for Windows XP/7, `/usr/local/apache-tomcat-5` for Linux/UNIX).
3. Setting `CATALINA_HOME` and `CLASSPATH` environment variables pointing to these locations.
4. Start Apache Tomcat.

Note: After successful startup, the default web applications included with Tomcat will be available at: http://localhost:8080/.

An example of web server set up: Tomcat

Setting `CATALINA_HOME` and `CLASSPATH` environment variables:

- Windows XP/7 – `set CATALINA=C:\apache-tomcat-5`
  `set CLASSPATH=%CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%`
- Linux/UNIX – `setenv CATALINA=/usr/local/apache-tomcat-5`
  `setenv CLASSPATH $CATALINA/common/lib/jsp-api.jar:$CLASSPATH`

An example of web server set up: Tomcat

Setting `CATALINA_HOME` and `CLASSPATH` environment variables:

- Windows XP/7 – `set CATALINA=C:\apache-tomcat-5`
  `set CLASSPATH=%CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%`
- Linux/UNIX – `setenv CATALINA=/usr/local/apache-tomcat-5`
  `setenv CLASSPATH $CATALINA/common/lib/jsp-api.jar:$CLASSPATH`

Starting Apache Tomcat:

- Windows XP/7 – `C:\apache-tomcat-5\bin\startup.bat`
- Linux/UNIX – `/usr/local/apache-tomcat-5/bin/startup.sh`

An example of web server set up: Tomcat

Setting `CATALINA_HOME` and `CLASSPATH` environment variables:

- Windows XP/7 – `set CATALINA=C:\apache-tomcat-5`
  `set CLASSPATH=%CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%`
- Linux/UNIX – `setenv CATALINA=/usr/local/apache-tomcat-5`
  `setenv CLASSPATH $CATALINA/common/lib/jsp-api.jar:$CLASSPATH`

Starting Apache Tomcat:

- Windows XP/7 – `C:\apache-tomcat-5\bin\startup.bat`
- Linux/UNIX – `/usr/local/apache-tomcat-5/bin/startup.sh`

Stopping Apache Tomcat:

- Windows XP/7 – `C:\apache-tomcat-5\bin\shutdown`
- Linux/UNIX – `/usr/local/apache-tomcat-5/bin/shutdown.sh`

Features of JSP

- Supports tag based programming.

Features of JSP

- Supports tag based programming.
- Strong java programming is not required. So,it is suitable for non-java programmers.

Features of JSP

- Supports tag based programming.
- Strong java programming is not required. So,it is suitable for non-java programmers.
- Gives nine implicit objects that can be used directly with additional codes (to access them).

Features of JSP

- Supports tag based programming.
- Strong java programming is not required. So,it is suitable for non-java programmers.
- Gives nine implicit objects that can be used directly with additional codes (to access them).
- Allows to separate presentation logic (html/code) from business logic (java code) exceptional handling is optional.

Features of JSP

- Supports tag based programming.
- Strong java programming is not required. So,it is suitable for non-java programmers.
- Gives nine implicit objects that can be used directly with additional codes (to access them).
- Allows to separate presentation logic (html/code) from business logic (java code) exceptional handling is optional.
- Increases the readability of code because of tags.

Features of JSP

- Supports tag based programming.
- Strong java programming is not required. So,it is suitable for non-java programmers.
- Gives nine implicit objects that can be used directly with additional codes (to access them).
- Allows to separate presentation logic (html/code) from business logic (java code) exceptional handling is optional.
- Increases the readability of code because of tags.
- Modifications will be reflected without re-compilation and re-loading.

Features of JSP

- Supports tag based programming.
- Strong java programming is not required. So,it is suitable for non-java programmers.
- Gives nine implicit objects that can be used directly with additional codes (to access them).
- Allows to separate presentation logic (html/code) from business logic (java code) exceptional handling is optional.
- Increases the readability of code because of tags.
- Modifications will be reflected without re-compilation and re-loading.
- Gives built-in JSP tags and allows to develop custom JSP tags and to use third party supplied JSP tags.

Features of JSP

- Supports tag based programming.
- Strong java programming is not required. So,it is suitable for non-java programmers.
- Gives nine implicit objects that can be used directly with additional codes (to access them).
- Allows to separate presentation logic (html/code) from business logic (java code) exceptional handling is optional.
- Increases the readability of code because of tags.
- Modifications will be reflected without re-compilation and re-loading.
- Gives built-in JSP tags and allows to develop custom JSP tags and to use third party supplied JSP tags.
- Easy to learn and easy to apply.

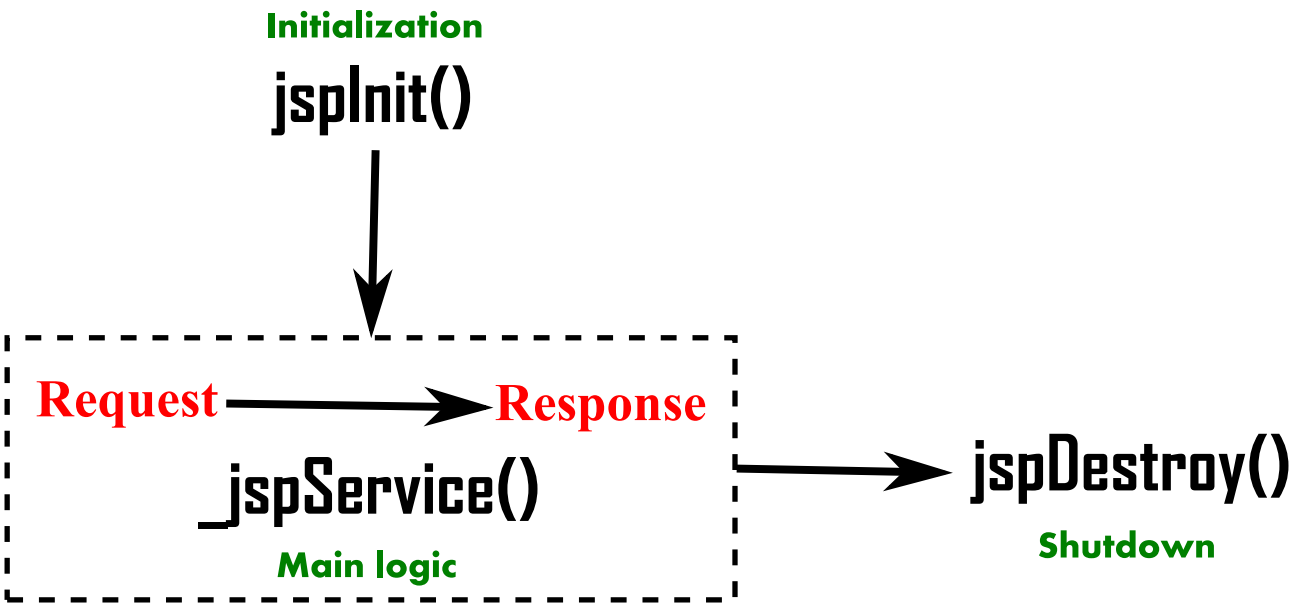
Features of JSP

- Supports tag based programming.
- Strong java programming is not required. So,it is suitable for non-java programmers.
- Gives nine implicit objects that can be used directly with additional codes (to access them).
- Allows to separate presentation logic (html/code) from business logic (java code) exceptional handling is optional.
- Increases the readability of code because of tags.
- Modifications will be reflected without re-compilation and re-loading.
- Gives built-in JSP tags and allows to develop custom JSP tags and to use third party supplied JSP tags.
- Easy to learn and easy to apply.
- Gives all the features of Servlet programming.

An interesting fact about JSP

- A JSP compiler is a program that parses JSPs, and transforms them into executable Java Servlets.

Life cycle of a JSP code



State transition diagram

Writing the first JSP program

Source: Welcome2JSP.jsp

Writing the first JSP program

Source: Welcome2JSP.jsp

```
<html>
    <head>
        <title>A Web Page</title>
    </head>
    <body>
        <% out.println("Welcome to JSP!"); %>
    </body>
</html>
```

Delimiters

JSP pages use several delimiters for scripting functions:

- The most basic delimiter is <% ... %>, which encloses a JSP scriptlet. A scriptlet is a fragment of Java code that is run when the user requests the page.

Delimiters

JSP pages use several delimiters for scripting functions:

- The most basic delimiter is <% ... %>, which encloses a JSP scriptlet. A scriptlet is a fragment of Java code that is run when the user requests the page.
- The delimiter <%= ... %> is used for expressions, where the value of the expression is placed into the page delivered to the user. They are mainly used for passing parameters.

Delimiters

JSP pages use several delimiters for scripting functions:

- The most basic delimiter is <% ... %>, which encloses a JSP scriptlet. A scriptlet is a fragment of Java code that is run when the user requests the page.
- The delimiter <%= ... %> is used for expressions, where the value of the expression is placed into the page delivered to the user. They are mainly used for passing parameters.
- The directives, represented as <%@ ... %>, are another special class of delimiters. They are mainly used to link with other pages.

Accessing dates

```
<html>
    <head>
        <title> A Date Test </title>
    </head>
    <body>
        <p> Today's date:  <%= (new
java.util.Date()).toLocaleString()%> </p>
    </body>
</html>
```

Branching

```
 <p>Binary decision</p>
<% int i = 10; %>
<% if(i>=0){ %>
    <p>Non-negative</p>
<%}%>
<% else{ %>
    <p>Negative</p>
<%}%>
<p>DONE</p>
```

Looping

```
 <p>First ten odd numbers:</p>
<% for (i=0; i<10; i=i+1){ %>
    <p>No.  <%= i+1 %>:  <%= 2*i+1 %>.</p>
<%}%>
<p>DONE</p>
```

Comments

Single-line java comment:

```
<% // This is a java message %>
```

Comments

Single-line java comment:

```
<% // This is a java message %>
```

Hidden comment:

```
<%-- This is hidden message --%>
```

Comments

Single-line java comment:

```
<% // This is a java message %>
```

Hidden comment:

```
<%-- This is hidden message --%>
```

Output comment:

```
<!-- This is output message >
```

Comments

Single-line java comment:

```
<% // This is a java message %>
```

Hidden comment:

```
<%-- This is hidden message --%>
```

Output comment:

```
<!-- This is output message >
```

Note: Comments cannot be nested

Objects in JSP

Object	Description
out	This is the `JspWriter` object used to send output to the client
session	This is the `HttpSession` object associated with the request
request	This is the `HttpServletRequest` object associated with the request
response	This is the `HttpServletResponse` object associated with the response to the client
application	This is the `ServletContext` object associated with application context
config	This is the `ServletConfig` object associated with the page
pageContext	This encapsulates use of server-specific features like higher performance JspWriters
page	This is simply a synonym for this, and is used to call the methods defined by the translated servlet class
Exception	This is the `Throwable` object that allows the exception data to be accessed by designated JSP

Using objects in JSP – `out`

JSP `out` implicit object is an instance of `javax.servlet.jsp.JspWriter` implementation and it's used to output content to be sent in client response.

Using objects in JSP – out

JSP `out` implicit object is an instance of `javax.servlet.jsp.JspWriter` implementation and it's used to output content to be sent in client response.

For writing any data to the buffer, JSP provides the object `out`, which is the object of JspWriter.

Using objects in JSP – out

JSP out implicit object is an instance of javax.servlet.jsp.JspWriter implementation and it's used to output content to be sent in client response.

For writing any data to the buffer, JSP provides the object out, which is the object of JspWriter.

```
 <html>
<body>
<% out.print("Today
is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Using objects in JSP – `session`

JSP `session` implicit object is an instance of `javax.servlet.http.HttpSession` implementation. Whenever we request a JSP page, container automatically creates a session for the JSP in the service method.

Using objects in JSP – session

JSP `session` implicit object is an instance of `javax.servlet.http.HttpSession` implementation. Whenever we request a JSP page, container automatically creates a session for the JSP in the service method.

Since session management is heavy process, so if we don't want session to be created for JSP, we can use page directive to not create the session for JSP using the following.

```
<%@ page session="false" %>.
```

Using objects in JSP – `session`

JSP `session` implicit object is an instance of `javax.servlet.http.HttpSession` implementation. Whenever we request a JSP page, container automatically creates a session for the JSP in the service method.

Since session management is heavy process, so if we don't want session to be created for JSP, we can use page directive to not create the session for JSP using the following.

```
<%@ page session="false" %>.
```

This is very helpful when our login page or the index page is a JSP page and we don't need any user session there.

Using objects in JSP – `request`

JSP `request` implicit object is an instance of `javax.servlet.http.HttpServletRequest` implementation and it is one of the argument of JSP service method.

Using objects in JSP – `request`

JSP `request` implicit object is an instance of `javax.servlet.http.HttpServletRequest` implementation and it is one of the argument of JSP service method.

We can use `request` object to get the request parameters, cookies, request attributes, session, header information and other details about client request.

Using objects in JSP – `response`

JSP `response` implicit object is an instance of `javax.servlet.http.HttpServletResponse` implementation and comes as the argument of service method.

Using objects in JSP – `response`

JSP `response` implicit object is an instance of `javax.servlet.http.HttpServletResponse` implementation and comes as the argument of service method.

We can use the `response` object to set the content type, character encoding, header information in response, adding cookies to response and redirecting the request to other resource.

Using objects in JSP – `application`

JSP `application` implicit object is an instance of `javax.servlet.ServletContext` implementation and it is used to get the context information and attributes in JSP.

Using objects in JSP – `application`

JSP `application` implicit object is an instance of `javax.servlet.ServletContext` implementation and it is used to get the context information and attributes in JSP.

We can use it to get the RequestDispatcher object in JSP to forward the request to another resource or to include the response from another resource in the JSP.

Using objects in JSP – `config`

JSP `config` implicit object is an instance of `javax.servlet.ServletConfig` implementation.

Using objects in JSP – `config`

JSP `config` implicit object is an instance of `javax.servlet.ServletConfig` implementation.

It is used to get the JSP init params (parameters) configured in deployment descriptor.

Using objects in JSP – pageContext

JSP `pageContext` implicit object is an instance of `javax.servlet.jsp.PageContext` abstract class implementation.

Using objects in JSP – pageContext

JSP `pageContext` implicit object is an instance of `javax.servlet.jsp.PageContext` abstract class implementation.

It is used to get and set the attributes with different scopes and to forward request to other resources. pageContext object also hold reference to other implicit object.

Using objects in JSP – page

JSP page implicit object is an instance of `java.lang.Object` class and represents the current JSP page.

Using objects in JSP – page

JSP page implicit object is an instance of `java.lang.Object` class and represents the current JSP page.

It provides reference to the generated servlet class. This object is very rarely used.

Using objects in JSP – `Exception`

JSP `exception` implicit object is an instance of `java.lang.Throwable` class and used to provide exception details in JSP error pages.

Using objects in JSP – `Exception`

JSP `exception` implicit object is an instance of `java.lang.Throwable` class and used to provide exception details in JSP error pages.

It can not be used in normal JSP pages and it is available only in JSP error pages.

Basics

Typical uses for HTTP Servlets include:

1. Processing and/or storing data submitted by an HTML form.
2. Providing dynamic content, e.g. returning the results of a database query to the client.
3. Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

Writing the first Servlet program

Source: TestServlet.java

Writing the first Servlet program

Source: TestServlet.java

```
  import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class TestServlet extends HttpServlet{
    public void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException,
ServletException{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>A Web Page</title>");
        out.println("</head>");
        out.println("</html>");
    }
}
```

Assignment

1. Write one JSP code and another java code using Servelet to separately create two tables in the server side.

Java and Web Technology

PHP

Malay Bhattacharyya

Assistant Professor

Department of Computer Science & Engineering
University of Kalyani, Nadia, West Bengal, India

May, 2014

1 Basics
- What is PHP?
- Features of PHP

2 Programming in PHP
- Writing the first PHP program
- Data types and variables
- Arrays
- Operators
- Functions
- In-built functions
- Branching
- Looping
- File handling
- Comments
- Handling MySQL with PHP

3 Assignment

What is PHP?

PHP is a server-side scripting language

What is PHP?

PHP is a server-side scripting language

The term PHP came from "Personal Home Page"

What is PHP?

PHP is a server-side scripting language

The term PHP came from "Personal Home Page"

1994: CGI developed independently by Rasmus Lerdorf for his personal homepage

What is PHP?

PHP is a server-side scripting language

The term PHP came from "Personal Home Page"

1994: CGI developed independently by Rasmus Lerdorf for his personal homepage
1995: Release of official version PHP 1.0

What is PHP?

PHP is a server-side scripting language

The term PHP came from "Personal Home Page"

1994: CGI developed independently by Rasmus Lerdorf for his personal homepage
1995: Release of official version PHP 1.0
1997: Zeev Suraski and Andi Gutmans rewrote the parser to produce PHP 3.0

What is PHP?

PHP is a server-side scripting language

The term PHP came from "Personal Home Page"

1994: CGI developed independently by Rasmus Lerdorf for his personal homepage
1995: Release of official version PHP 1.0
1997: Zeev Suraski and Andi Gutmans rewrote the parser to produce PHP 3.0

Note: Scripting language is a high-level programming language that is interpreted by another program at runtime.

Features of PHP

- It is a scripting language.

Features of PHP

- It is a scripting language.
- It is an interpreted language.

Features of PHP

- It is a scripting language.
- It is an interpreted language.
- It is a functional programming language.

Features of PHP

- It is a scripting language.
- It is an interpreted language.
- It is a functional programming language.
- It supports object-based programming.

Some facts about PHP

- PHP is an open source software and therefore free to download and use.

Some facts about PHP

- PHP is an open source software and therefore free to download and use.
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.).

Some facts about PHP

- PHP is an open source software and therefore free to download and use.
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.).
- PHP runs on different platforms (Windows, Linux, Unix, etc.).

Some facts about PHP

- PHP is an open source software and therefore free to download and use.
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.).
- PHP runs on different platforms (Windows, Linux, Unix, etc.).
- PHP is compatible with almost all servers used today (Apache, IIS, etc.).

Some facts about PHP

- PHP is an open source software and therefore free to download and use.
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.).
- PHP runs on different platforms (Windows, Linux, Unix, etc.).
- PHP is compatible with almost all servers used today (Apache, IIS, etc.).
- PHP code is partially case sensitive (only variables, constants, array keys, class properties and class constants).

Some facts about PHP

- PHP is an open source software and therefore free to download and use.
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.).
- PHP runs on different platforms (Windows, Linux, Unix, etc.).
- PHP is compatible with almost all servers used today (Apache, IIS, etc.).
- PHP code is partially case sensitive (only variables, constants, array keys, class properties and class constants).
- PHP is a loosely typed language.

Writing the first PHP program

<u>Source</u>: Welcome2PHP.php

Writing the first PHP program

Source: Welcome2PHP.php

```
 <html>
<title>Entering into PHP</title>
<body>
<?php
echo ‘Welcome to PHP’;
?>
</body>
</html>
```

Writing the first PHP program

Source: Welcome2PHP.php

```
 <html>
<title>Entering into PHP</title>
<body>
<?php
echo ‘Welcome to PHP’;
?>
</body>
</html>
```

Note: PHP files can also have the extensions “.php3” and “.phtml”.

Data types and variables

PHP does not need to declare a data type for a variable

Data types and variables

PHP does not need to declare a data type for a variable

```
 $pi = 3.14 // variable pi is assigned a real
constant
$batch = "MCA"; // variable batch is assigned a
string
```

Data types and variables

PHP does not need to declare a data type for a variable

```
 $pi = 3.14 // variable pi is assigned a real
constant
$batch = "MCA"; // variable batch is assigned a
string
```

Note: A variable name can only contain alphanumeric characters and underscores (a-z, A-Z, 0-9, and _), and must start with a letter or an underscore "_".

Arrays

```
 $TopBikes =
array("BMW","Ducati","Yamaha","Kawasaki","Honda");
```

Arrays

```
 $TopBikes =
array("BMW","Ducati","Yamaha","Kawasaki","Honda");

$TopCars = array();
$TopCars[0] = "Bugatti";
$TopCars[1] = "Ferrari";
$TopCars[2] = "Porsche";
```

Arrays

```
 $TopBikes =
array("BMW","Ducati","Yamaha","Kawasaki","Honda");

$TopCars = array();
$TopCars[0] = "Bugatti";
$TopCars[1] = "Ferrari";
$TopCars[2] = "Porsche";

$NumberArray = array(
array(0, 1, 2),
array(3, 4, 5),
);
```

Operators

Arithmetic operators: `+`, `-`, `*`, `/`, `%`
Relational operators: `<`, `<=`, `>`, `>=`, `==`, `!=`, `===` (equality without type conversion)
Logical operators: `&&`, `||`, `!`
Assignment operators: `=`
Increment and decrement operators: `++`, `--`
Special operators: `.`

Operators

Arithmetic operators: `+`, `-`, `*`, `/`, `%`
Relational operators: `<`, `<=`, `>`, `>=`, `==`, `!=`, `===` (equality without type conversion)
Logical operators: `&&`, `||`, `!`
Assignment operators: `=`
Increment and decrement operators: `++`, `--`
Special operators: `.`

Note: The special operator '.' is used to concatenate two strings.

Functions

```
 <?php
function myFunction(){
    echo "Welcome to KU";
}
?>
```

In-built functions

Length of a string:

```
<?php
$S1 = "Hunny";
$S2 = "Bunny";
echo $S1 . " " . $S2 . " " . strlen($S1);
?>
```

In-built functions

Length of a string:

```
<?php
$S1 = "Hunny";
$S2 = "Bunny";
echo $S1 . " " . $S2 . " " . strlen($S1);
?>
```

Finding a substring in a string:

```
<?php
echo strpos("Slovenia","love");
?>
```

In-built functions

Length of a string:

```
<?php
$S1 = "Hunny";
$S2 = "Bunny";
echo $S1 . " " . $S2 . " " . strlen($S1);
?>
```

Finding a substring in a string:

```
<?php
echo strpos("Slovenia","love");
?>
```

Note: The string position always starts from a zero.

In-built functions

Accessing IP address of the user:

```
<?php
$IP = $_SERVER['REMOTE_ADDR'];
echo $IP;
?>
```

Branching

Single branching:

```
<?php
$d = date("D");
if($d=="Fri") echo "Have a nice weekend!";
?>
```

Branching

Single branching:

```
<?php
$d = date("D");
if($d=="Fri") echo "Have a nice weekend!";
?>
```

Nested branching:

```
<?php
$d = date("D");
if($d=="Fri")
    echo "Have a nice weekend!";
elseif($d == "Sun")
    echo "Have a nice Sunday!";
else
    echo "Have a nice day!";
?>
```

Branching

```
 <?php
$position = 2;
switch($position){
case 1:
    echo "First";
    break;
case 2:
    echo "Second";
    break;
case 3:
    echo "Third";
    break;
default:
    echo "No ranks";
}
?>
```

Looping

```
 <?php
$i=1;
while($i<=5){
    echo "The number is " .  $i .  "<br />";
$i++;
}
?>
```

File handling

Reading from a file:

```
<?php
$file = fopen('Students.txt', 'r');
$Header = fgets($file); // Read the header line
fclose($file);
?>
```

File handling

Reading from a file:

```
<?php
$file = fopen('Students.txt', 'r');
$Header = fgets($file); // Read the header line
fclose($file);
?>
```

Writing to a file:

```
<?php
$file = fopen('Result.txt', 'w');
fwrite($file, 'Nobody failed!');
fclose($file);
?>
```

Comments

Single-line comment:

// ...

Comments

Single-line comment:

// ...

Multi-line comment:

/* ...

...

... */

Comments

Single-line comment:

// ...

Multi-line comment:

/* ...

...

... */

Shell-style comment:

...

Comments

Single-line comment:

// ...

Multi-line comment:

/* ...

...

... */

Shell-style comment:

...

Note: Comments cannot be nested

Handling MySQL with PHP

Verifying MySQL connection:

```
<?php
echo mysql_stat();
?>
```

Handling MySQL with PHP

Verifying MySQL connection:

```
<?php
echo mysql_stat();
?>
```

Connecting to a local host:

```
<?php
$conn = mysql_connect('localhost');
echo $conn;
?>
```

Handling MySQL with PHP

Creating a database:

```
 <?php
mysql_connect('localhost');
if(mysql_query("CREATE DATABASE KU")){
    echo "Database created ...";
}
else{
    echo mysql_error();
}
?>
```

Handling MySQL with PHP

Creating a table:

```
<?php
mysql_select_db('MCA');
$result = mysql_query("CREATE TABLE users(id
varchar(16) binary NOT NULL default '', Password
varchar(16) NOT NULL default '', Name varchar(64)
default NULL, email varchar(64) default NULL, ts
timestamp(14) NOT NULL, PRIMARY KEY (id))");
if($result){
    echo "Table created";
}
else{
    echo mysql_error();
}
?>
```

Handling MySQL with PHP

Using the insert query:

```
  <?php
function add_user($id, $pass, $name, $email){
    $result = mysql_query("insert into users values ('$id',
ENCRYPT('$pass'), '$name', '$email', NULL)");
    if($result){
        echo "Row inserted<br />";
    }
    else{
        echo mysql_error()."<br />";
    }
}
mysql_connect('localhost');
mysql_select_db('KU');
add_user('S01', 'PM123', 'Piyali Mondal',
'piyali@klyuniv.ac.in');
add_user('S02', 'Aich123', 'Atanu Aich', 'atanu@klyuniv.ac.in');
?>
```

Assignment

1. Write a PHP code that will show a message on pressing a button.