Introduction to Programming

Java – Built-in Classes, Built-in Methods, File Programming

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

February, 2021

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?

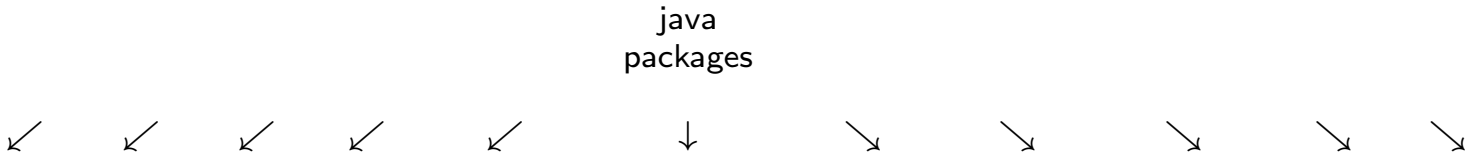
With packages

Basics

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

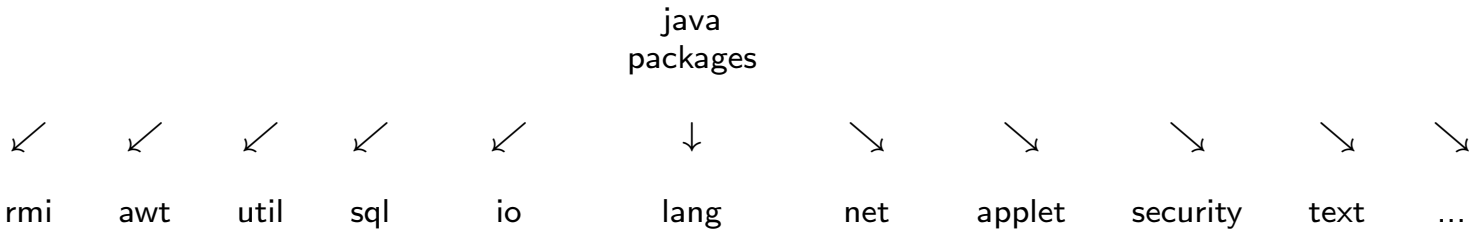
Basics

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



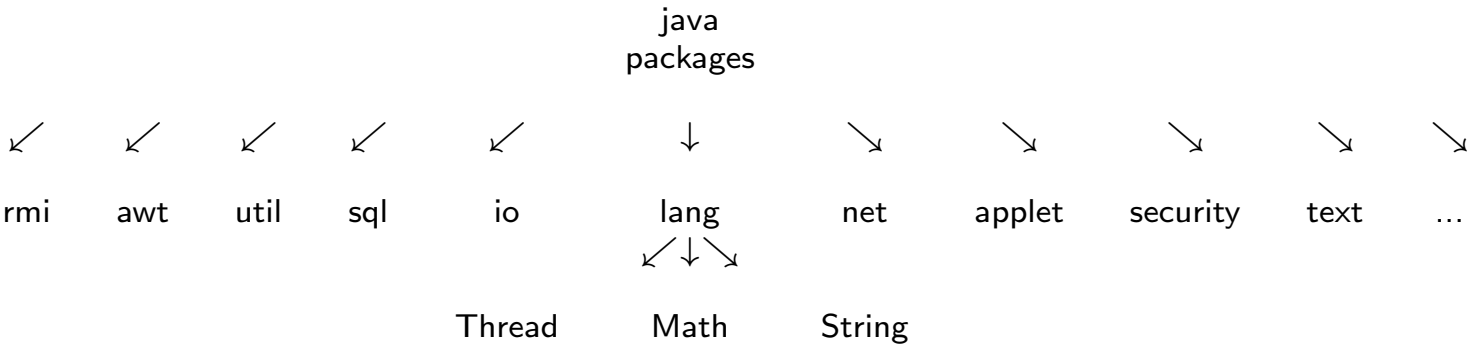
Basics

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



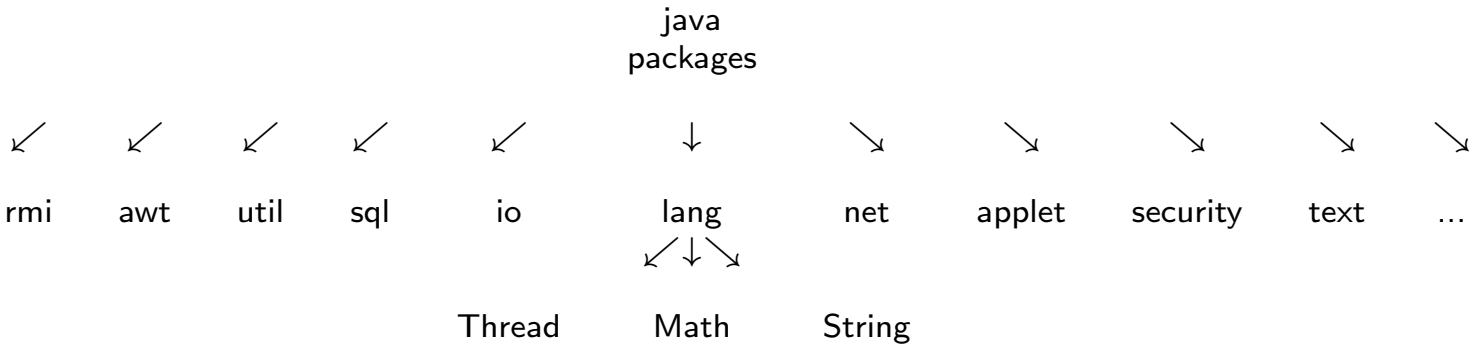
Basics

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

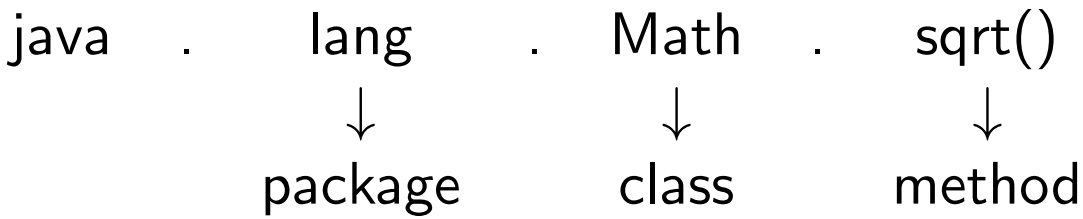


Note: Find more in books

Organization and use of a package

java . lang . Math . sqrt()

Organization and use of a package



Organization and use of a package

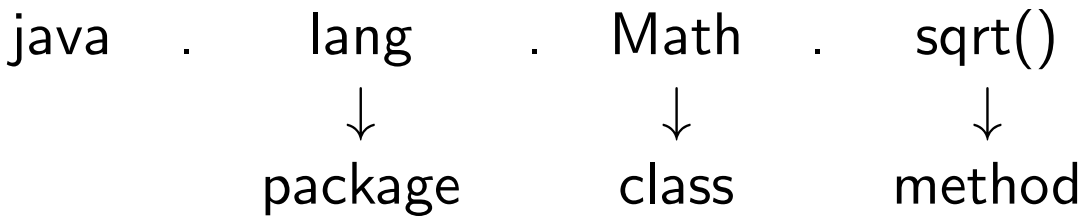
java . lang . Math . sqrt()

↓ ↓ ↓

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




```
 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.

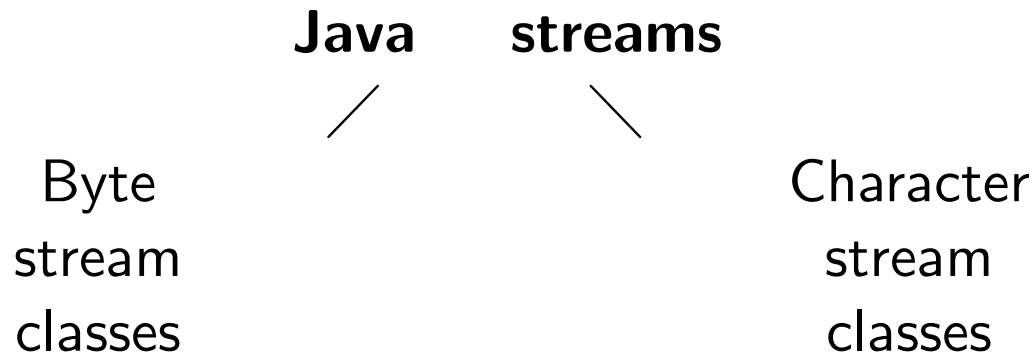
Concept of streams

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

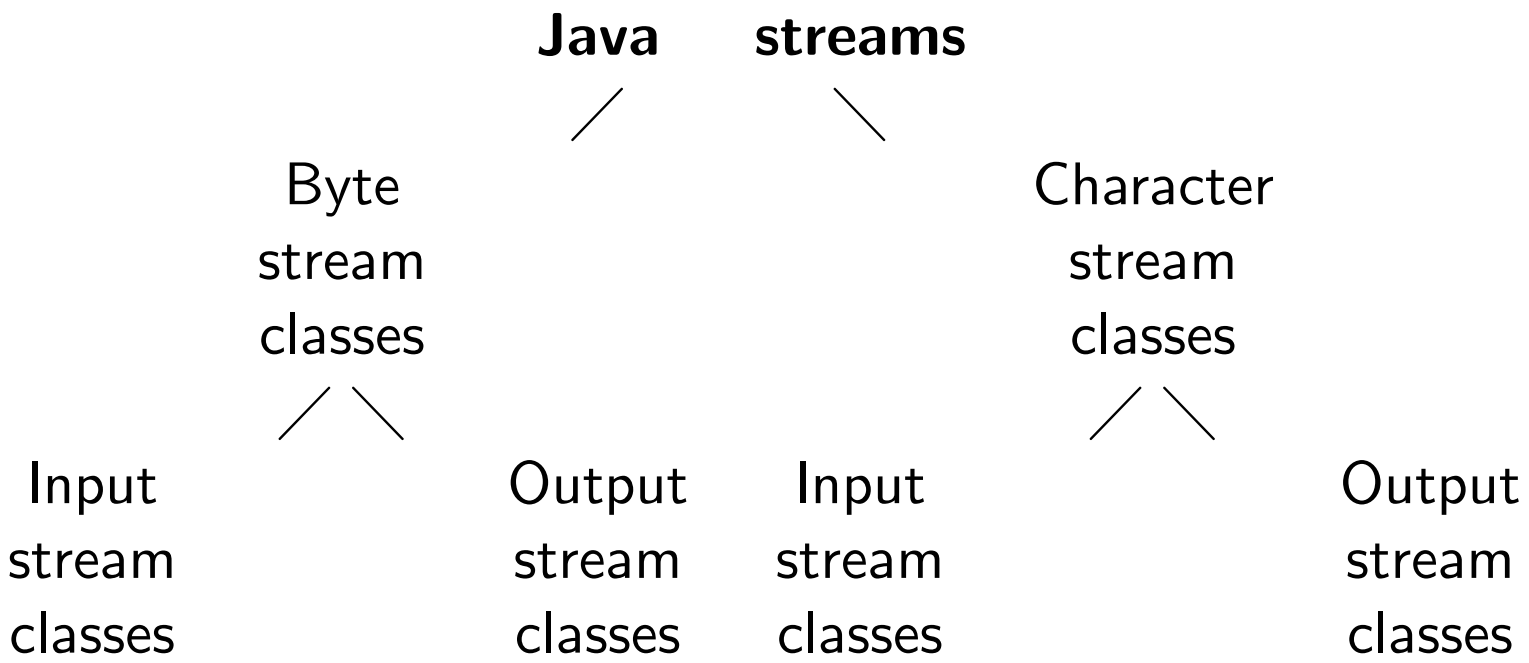
The stream classes in java

Java streams

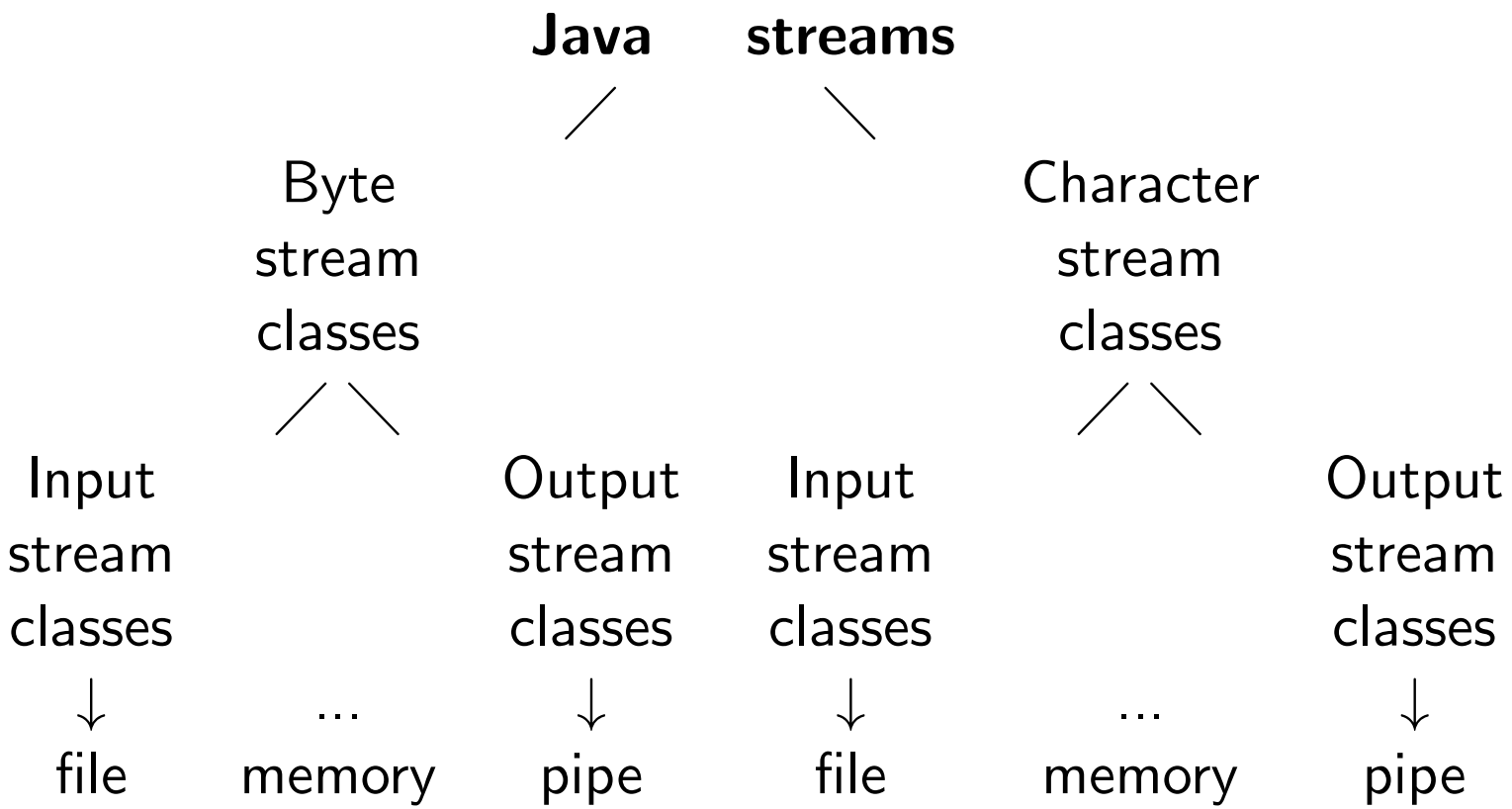
The stream classes in java



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();
    }
}
```