JAVA

Core JAVA

1. Introduction to Java
2. Java Features
3. Java Editors
4. JDK installation
5. First java program
6. classes and objects
7. new keyword
8. javac and java commands
9. Constructor and Constructor Overloading
10. instance and static variables
11. instance and static methods

12. instance and static block
13. javap command
14. Data-types
15. Literals
16. Operators
17. Conditional statements
18. Loops
19. Enhanced for loop
20. Command line arguments
21. Wrapper classes
22. Keyword this
    a. accessing instance variables and methods
    b. invoking current class constructors of different argument

23. Difference between constructors and methods
24. Principles of OOPs
    a. Encapsulation
    b. Inheritance
    c. Polymorphism
    d. Abstraction
25. Encapsulation
    a. Private fields
    b. Setter methods
    c. Getter methods
26. Inheritance
    a. Types of inheritance
    b. Constructor chain in inheritance

   c. Keyword super
      i. accessing super type variables and methods
      ii. invoking super type constructors of different argument
   d. Method Overriding
   e. Super type referencing the sub type objects
27. Polymorphism with method overloading and overriding
28. Abstraction
   a. Interface
   b. Marker Interface
29. Non Access modifiers like static, abstract and final
30. Abstract classes
31. Difference between abstract class and interface

32. Type Conversion
    a. auto widening and explicit narrowing
    b. auto upcasting and explicit downcasting
    c. instance operator
33. Keyword final
    a. For variables
    b. For classes
    c. For methods
34. Packages in Java
    a. create user-defined package
    b. importing the user-defined packages

35. Access specifier
    a. private scope
    b. package scope
    c. protected scope
    d. public scope
36. Scanner class
37. Exception Handling
    a. Keyword try
    b. Keyword catch
    c. Keyword finally
    d. Keyword throw
    e. Keyword throws
    f. Exception hierarchy

g. Checked and Unchecked exceptions
h. Handling all types of exceptions
i. Custom exception

38. Predefined packages
    a. java.lang
    b. java.io
    c. java.util
39. Multithreading
    a. Runnable interface
    b. Thread class
    c. start and run difference
    d. Some common methods in Thread
    e. Synchronization

40. Object class
41. String, StringBuffer and StringBuilder
    a. Difference between these classes
    b. Some useful methods
42. Math class
43. Runtime and Process class
44. Util Package Classes
    a. Date
    b. Stack
    c. StringTokenizer
    d. BitSet
    e. Scanner etc.

45. Collection Framework
    a. List
        i. ArrayList
        ii. Vector
        iii. LinkedList
    b. Set
        i. HashSet
        ii. LinkedHashSet
        iii. TreeSet
    c. Queue
        i. PriorityQueue
    d. Using Enumerator
    e. Using Iterator

46. Map
    a. HashTable
    b. HashMap
    c. LinkedHashMap
    d. TreeMap
47. Generics
    a. Advantages of generics over non genric Collection
    b. Applying generics for Collection APT’s
48. Collection class
49. Static import
50. Enums
51. Var-args

Introduction to JAVA

What is JAVA?

- JAVA is a Pure Object Oriented Programming Language developed by SunMicroSystems.
- Java is a Platform (JRE and API).

Platform: It’s any hardware or software environment in which a program runs.

Introduction to JAVA (contd.)

Few Languages before JAVA?

- **Machine Code**
- **Assembly Language**
- **C**
- **Smalltalk**
- **C++**
- **Java**

Introduction to JAVA (contd.)

History of JAVA

- Java was developed by James Gosling, Patrick Naughton and Mike Sheridan.
- It was initiated in 1991 and was called as Green.
- It was initially developed in 1993 and was initially named Oak.
- Finally in 1995 Java 1.0 was released.

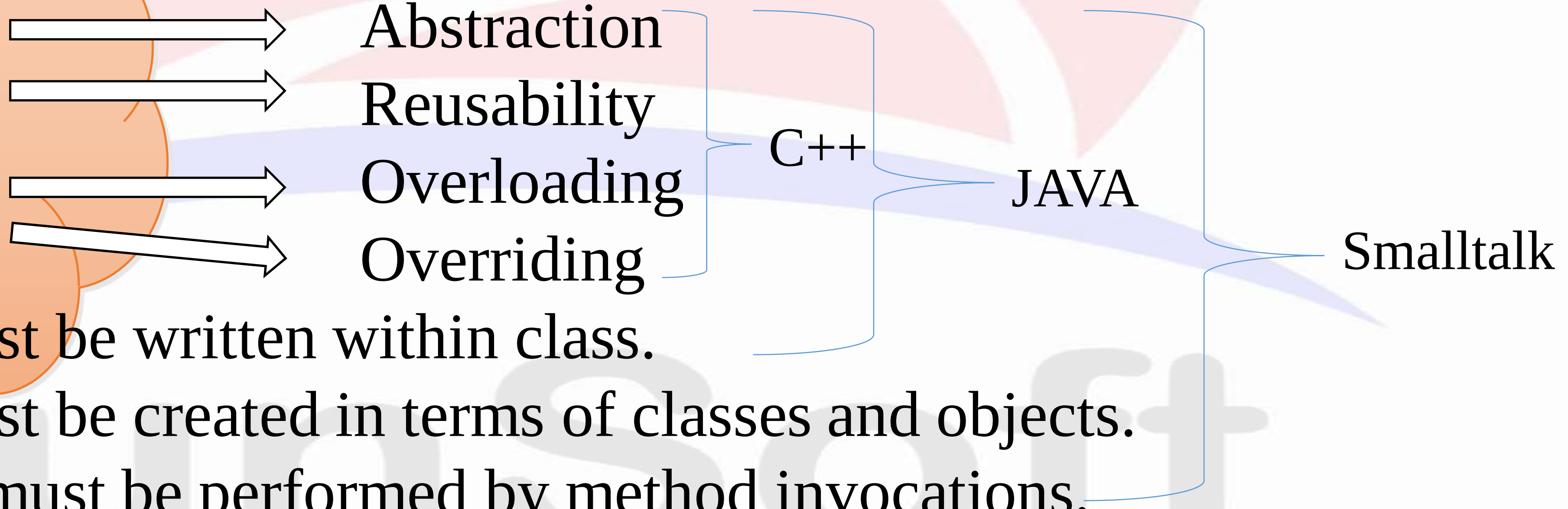
Introduction to JAVA (contd.)

*7	:	First project developed using Java (1993-mids)
HotJava	:	browser to advertise about Java (1995)
Netscape	:	gave Java it's first break (1996)

Introduction to JAVA (contd.)

Properties of OOP Language:

- Encapsulation ⟹ Abstraction
- Inheritance ⟹ Reusability
- Polymorphism ⟹ Overloading ⟹ Overriding
- Everything must be written within class.
- Everything must be created in terms of classes and objects.
- Every actions must be performed by method invocations.



Why java is called as Pure OOP language?

JAVA Features

- Simple
- Pure OOP
- Platform independent
- Portable
- Secure
- Robust
- Dynamic
- Multi-threaded
- Exception Handling
- Distributed
- Both compiled and interpreted

JAVA Editions

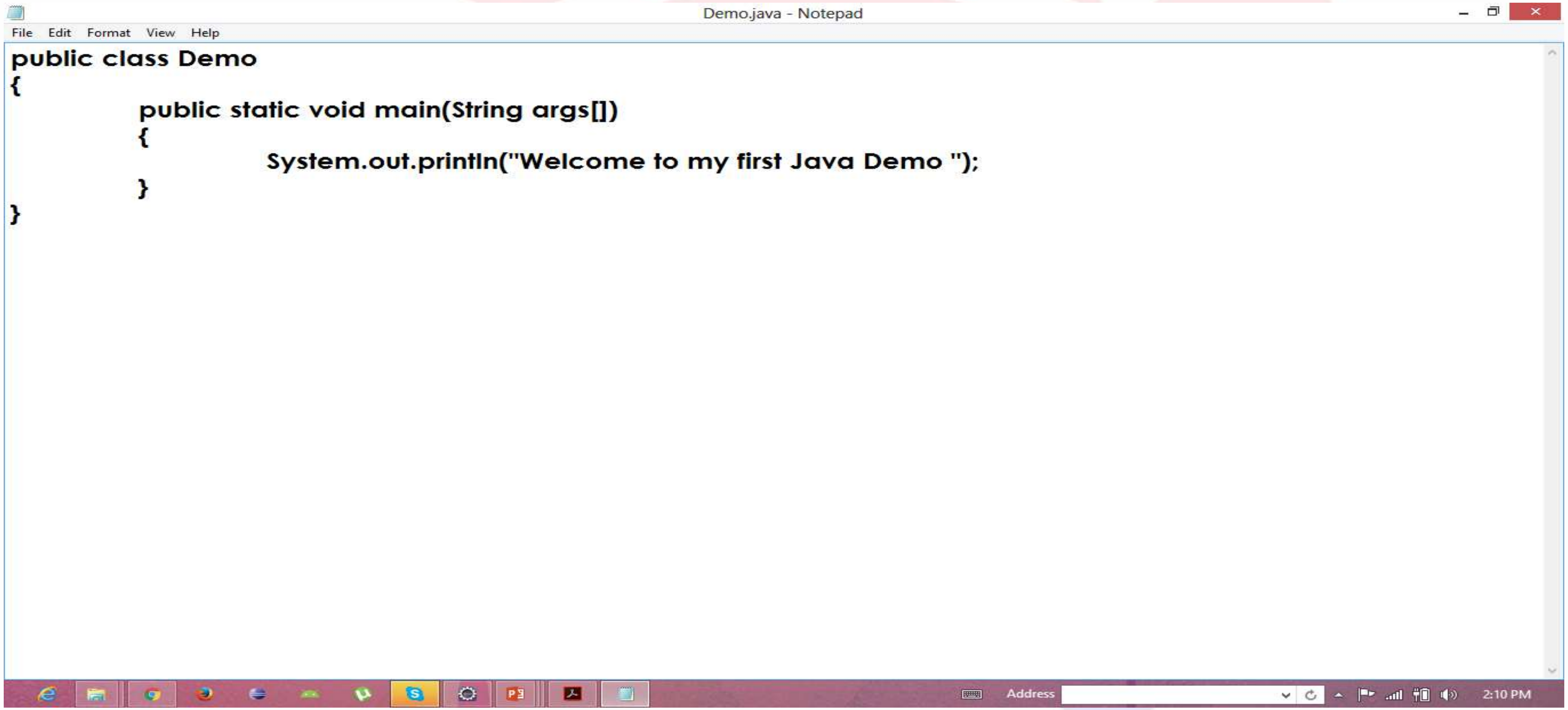
- Java SE
- Java EE
- Java ME

JDK Installation

JDK stands for Java Development Kit. It is required to develop Java programs.
To install JDK follow these simple steps:

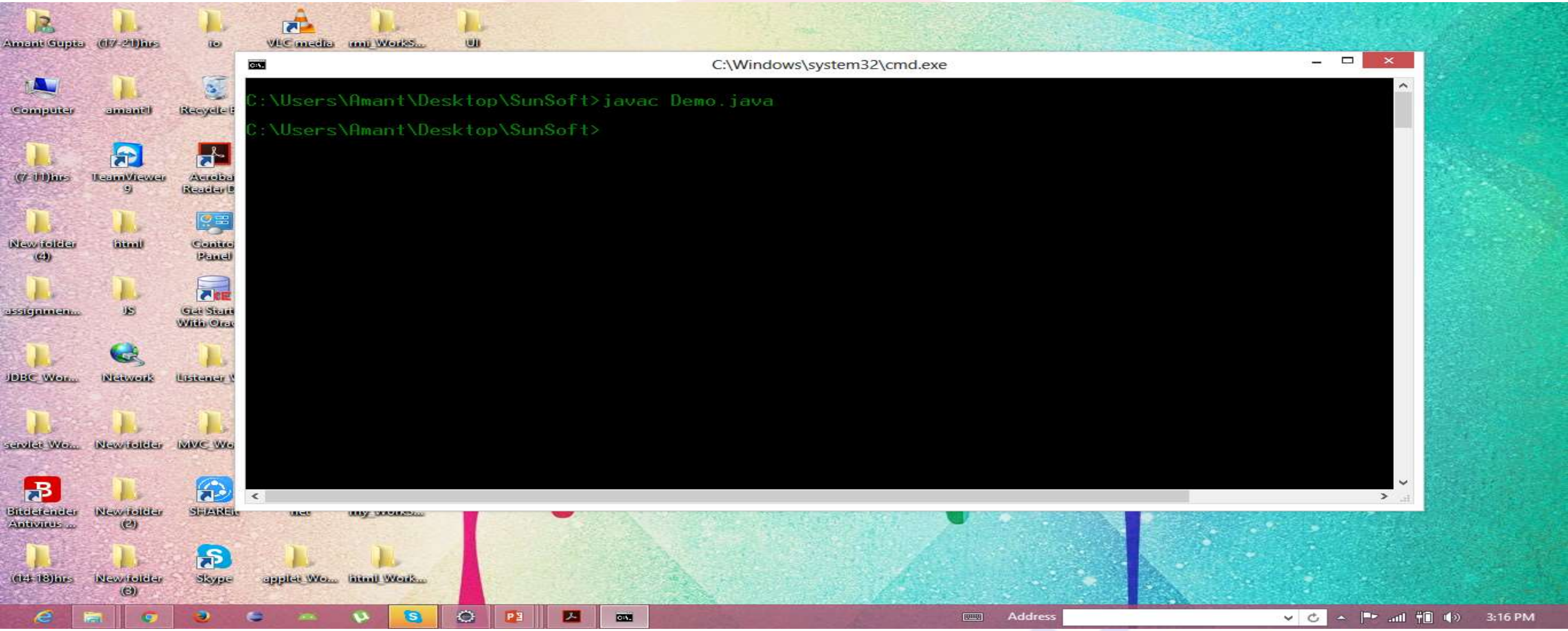
1. Download JDK
2. Install JDK and JRE
3. Include JDK’s “bin” Directory in the Path.
   - JDK’s “bin” Directory can be also used in “.bat” file.

First Java Program

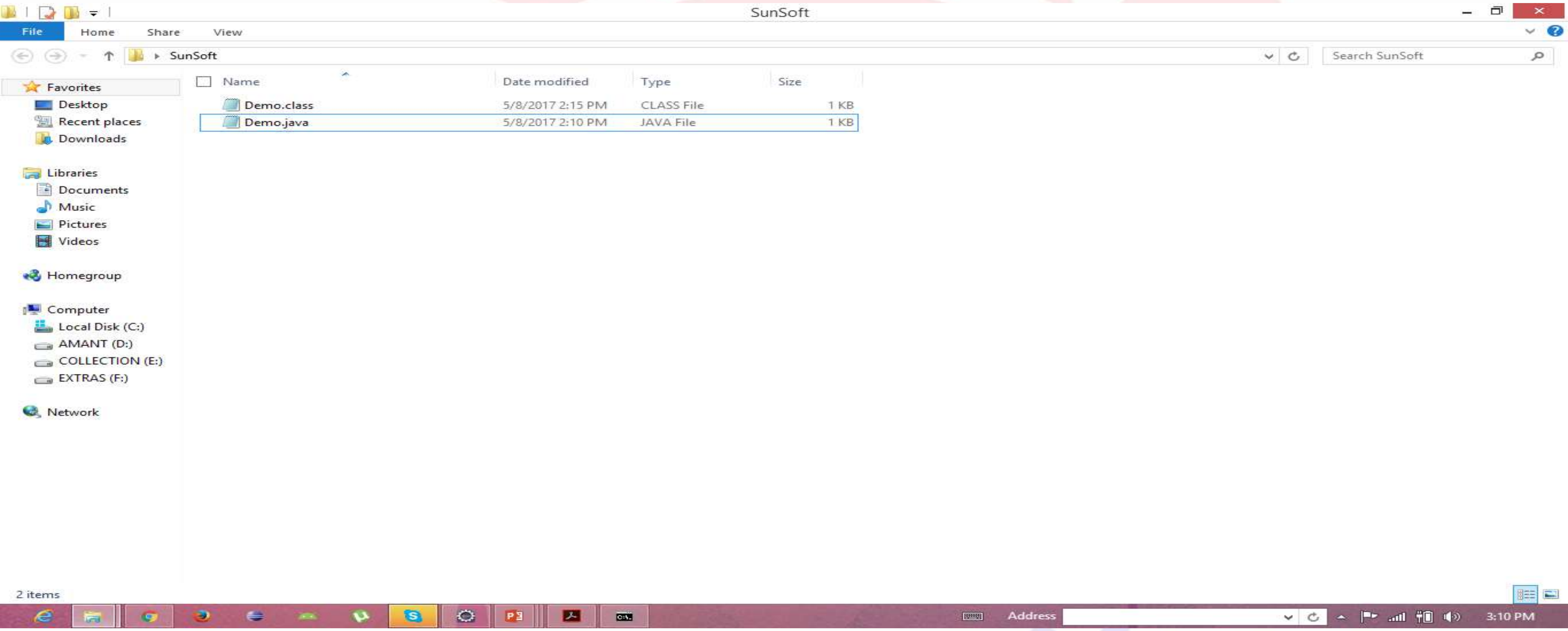


```
public class Demo
{
		public static void main(String args[])
		{
			System.out.println("Welcome to my first Java Demo ");
		}
}
```

First Java Program (contd.)


```
C:\Users\Amant\Desktop\SunSoft>javac Demo.java

C:\Users\Amant\Desktop\SunSoft>
```

First Java Program (contd.)

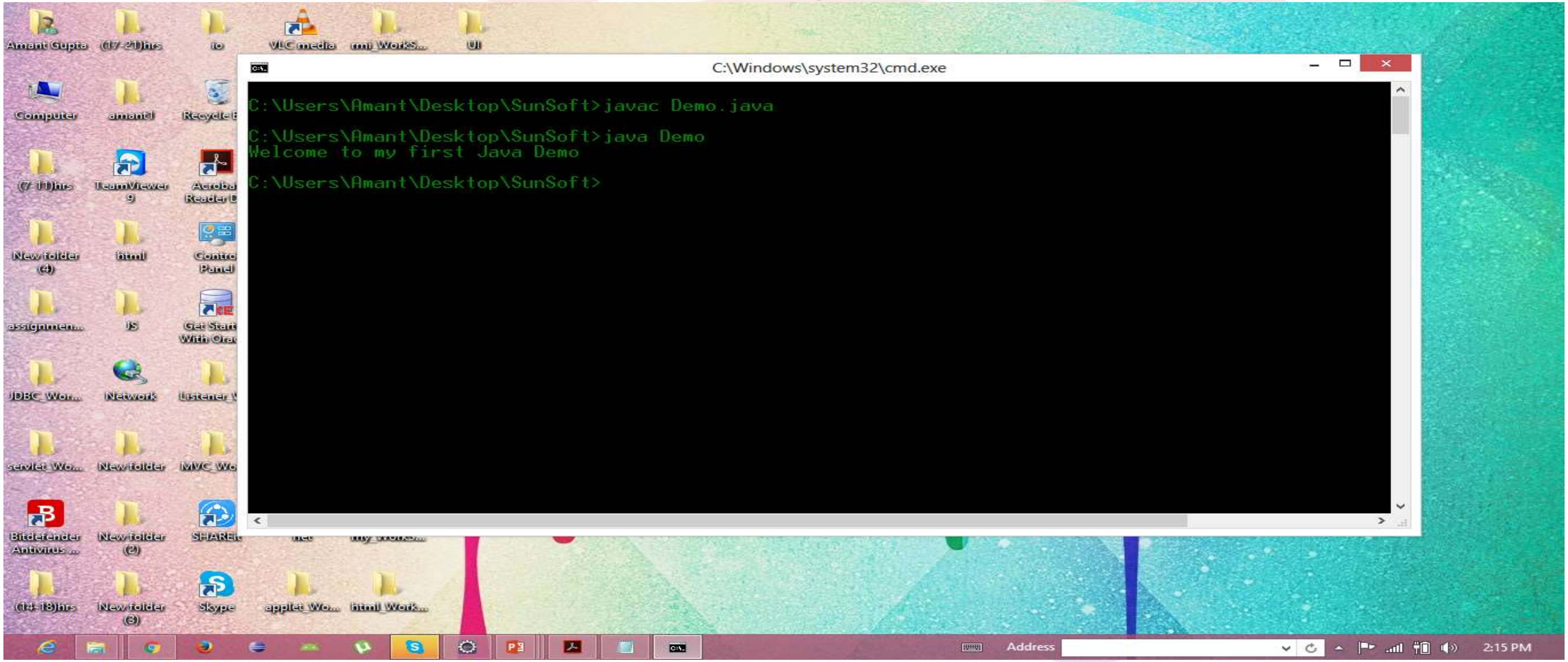


SunSoft

Name	Date modified	Type	Size
Demo.class	5/8/2017 2:15 PM	CLASS File	1 KB
Demo.java	5/8/2017 2:10 PM	JAVA File	1 KB

2 items

First Java Program Output



```
C:\Users\Amant\Desktop\SunSoft>javac Demo.java

C:\Users\Amant\Desktop\SunSoft>java Demo
Welcome to my first Java Demo

C:\Users\Amant\Desktop\SunSoft>
```

Setting the path

How to set path to execute a java program?

By setting the environment variable

By creating .bat file

Setting the path (contd.)

How to set the environment variable?

- ➢Right-click on ‘Computer’ icon.
- ➢Select ‘Properties’ option.
- ➢Click on the ‘Environment Variables’ button under the ‘Advanced’ tab.
- ➢Alter the ‘path’ variable.
  - You can also add a new variable.
- ➢Add ‘C:\Program Files\Java\jdk1.7.0\bin’ path to the existing path.
  - Provide a semicolon (;) after the default path.
- ➢Finally save it.

Setting the path (contd.)

How to create .bat file?

- To create .bat file open notepad.
- Then write “set path = PATH_OF_JDK-BIN then append ;.;%path%”
- Then write “set classpath = .;%classpath%”
- Finally write start.

Classes and Objects

Definitions:

- **Class** is a blueprint or template of an object.
  - E.g.: blueprint of a building

- **Object** is an instance of a class or any real world entity.
  - E.g.: building

Classes and Objects(contd.)

- Class name and file name should be same if that class is ‘public’.
- We can’t create more than one public class in a same file.
- ‘.java’ file is not converted to ‘.class’
- ‘.class’ file is made of every class regardless of ‘main method’ and ‘public class’.
- We can create more than one ‘main method’ in a single file.
- We can execute only that class which has ‘main method’.
- It’s a good practice to keep the main method in the class which has ‘business logic’.

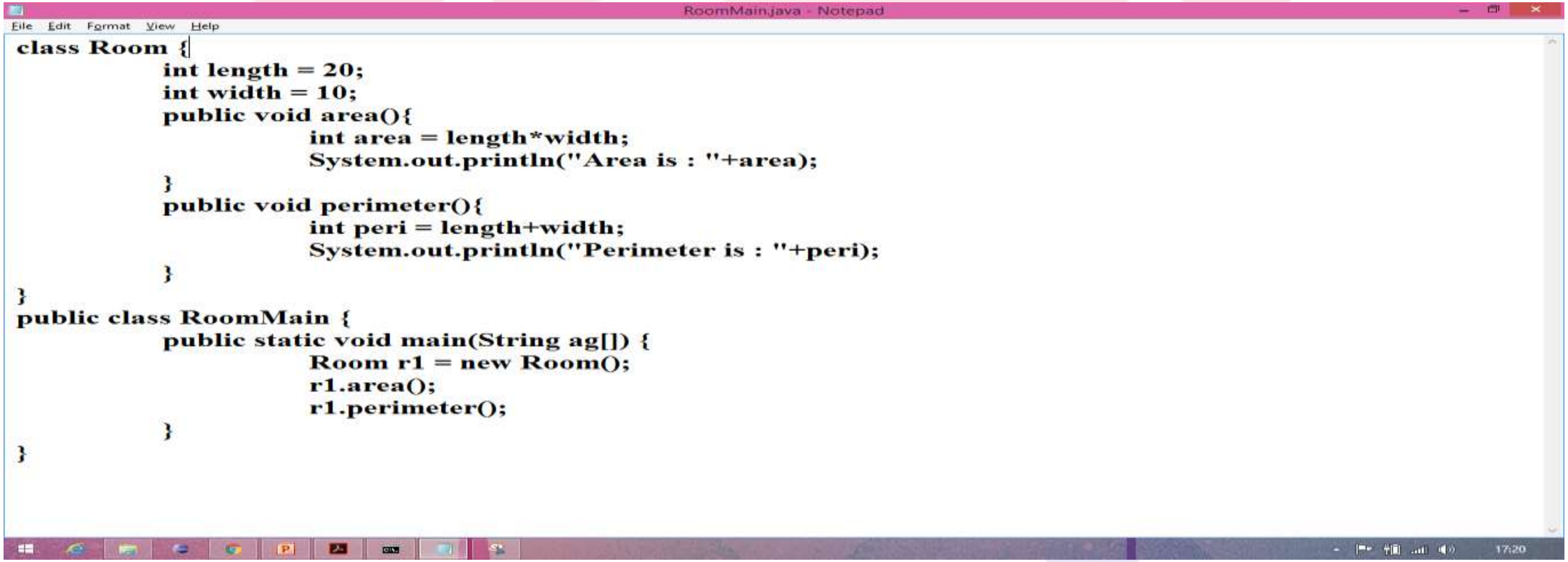
Classes and Objects (contd.)

Naming convention in Java (CamelCase)

Noun	ClassName
Verb	methodName
Entity	variable

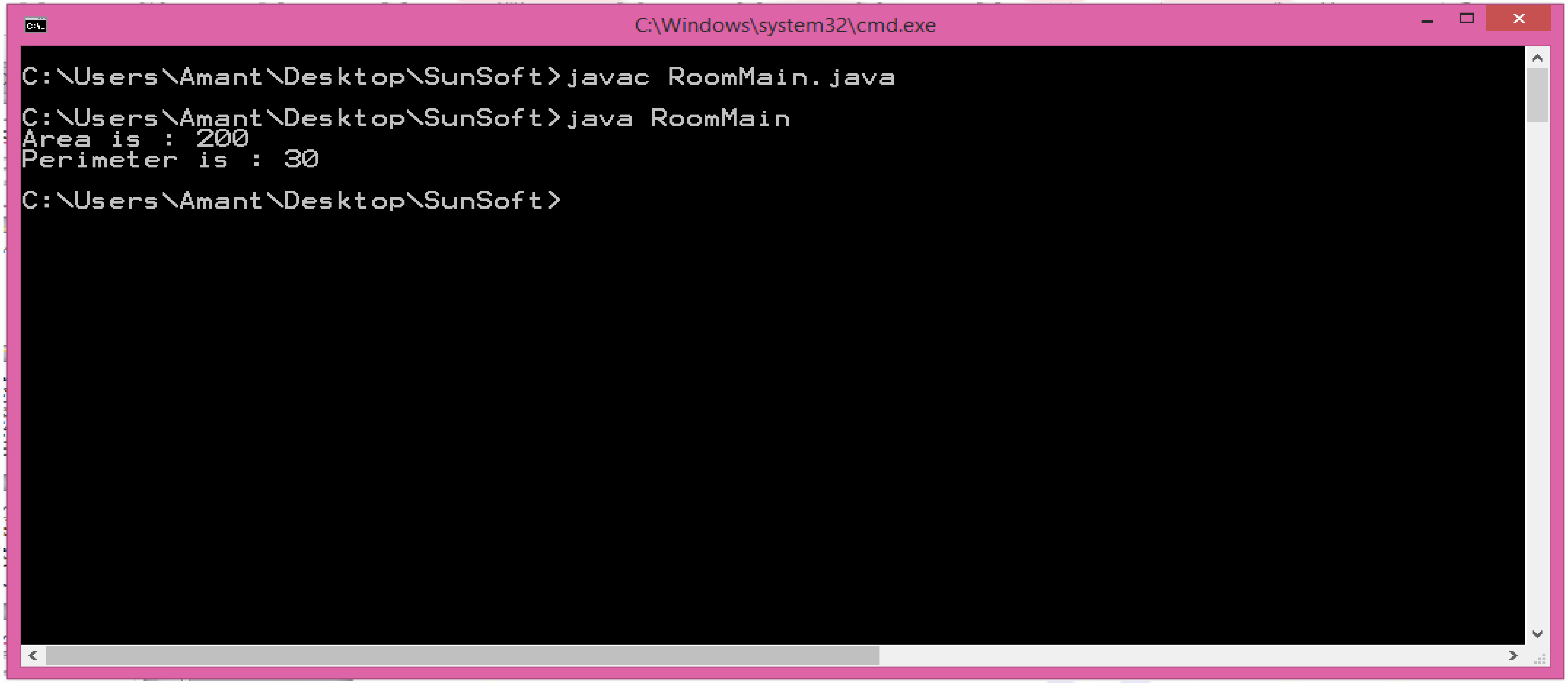
‘new’ keyword

- ‘new’ keyword creates the object of a class.
- ‘new’ keyword assigns memory when any object is created.



```
class Room {
		int length = 20;
		int width = 10;
		public void area(){
				int area = length*width;
				System.out.println("Area is : "+area);
		}
		public void perimeter(){
				int peri = length+width;
				System.out.println("Perimeter is : "+peri);
		}
}
public class RoomMain {
		public static void main(String ag[]) {
				Room r1 = new Room();
				r1.area();
				r1.perimeter();
		}
}
```

‘new’ keyword (contd.)



```
C:\Users\Amant\Desktop\SunSoft>javac RoomMain.java

C:\Users\Amant\Desktop\SunSoft>java RoomMain
Area is : 200
Perimeter is : 30

C:\Users\Amant\Desktop\SunSoft>
```

javac and java command

- javac is a component of JDK.
- It is used to compile the program i.e. it checks the syntactical errors and converts source code into byte code.
- java is a component of JDK as well as JRE.
- It is used to run the “.class’ file i.e. it converts byte code into machine code.

Constructors and overloading constructors

Constructors are a kind of special method.

- Constructors must have the same name as the class itself whereas methods can have any name including the class name.
- There is no need of calling constructors, unlike methods constructors are automatically called when we create object of the class.
- We can invoke methods ‘n’ number of times but constructors can be invoked only once i.e. during the creation of object.
- We can put any logic in constructors alike methods.
- Return type can’t be used with constructors.
- If return type(s) are used with constructors then it behaves like method.
- Non-access modifiers can’t be used with constructors.

Constructors and overloading constructors (contd.)

Default Constructor:

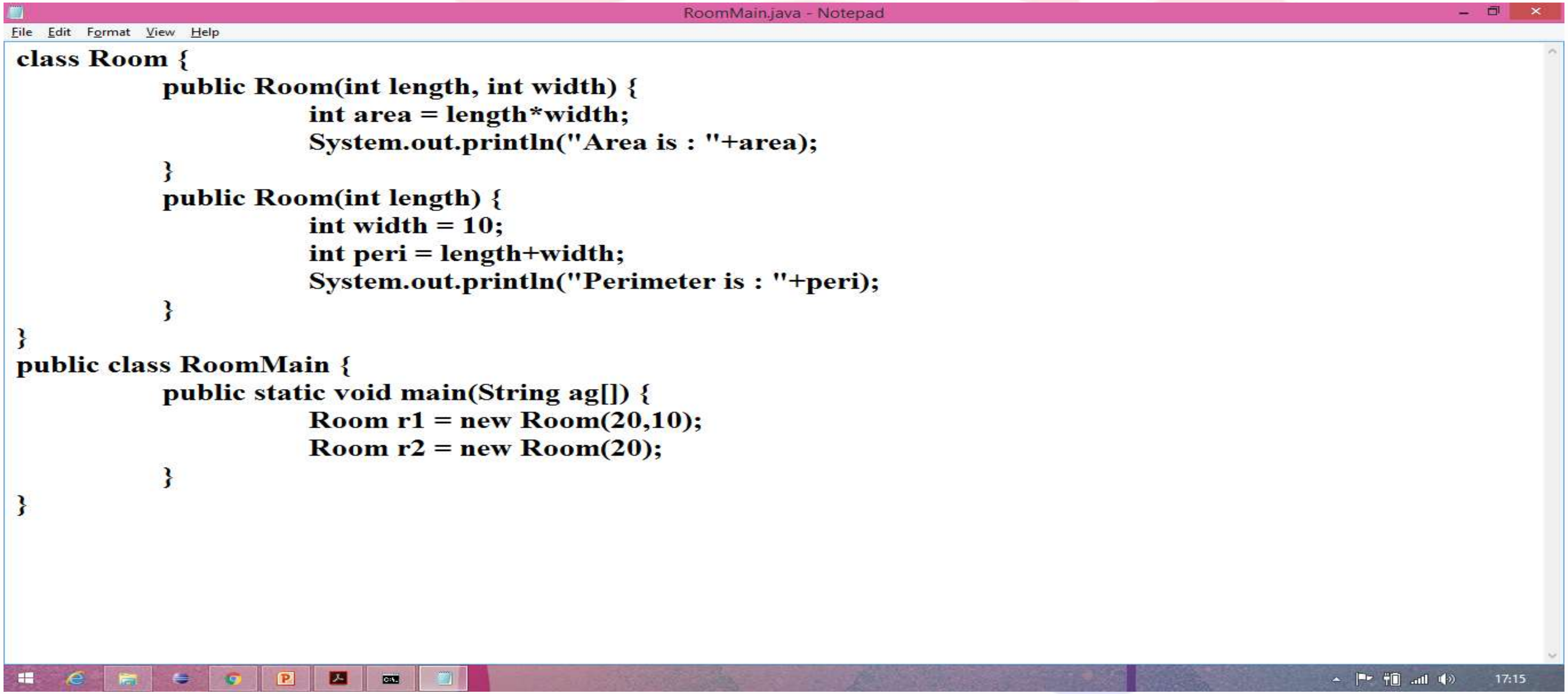
Default constructor is provided by compiler, which is always empty and if any constructor is declared explicitly then no default constructor is provided.

Constructor Overloading:

Constructor overloading is the phenomenon of creating more than one constructor with same name but with different parameter list in the same class.

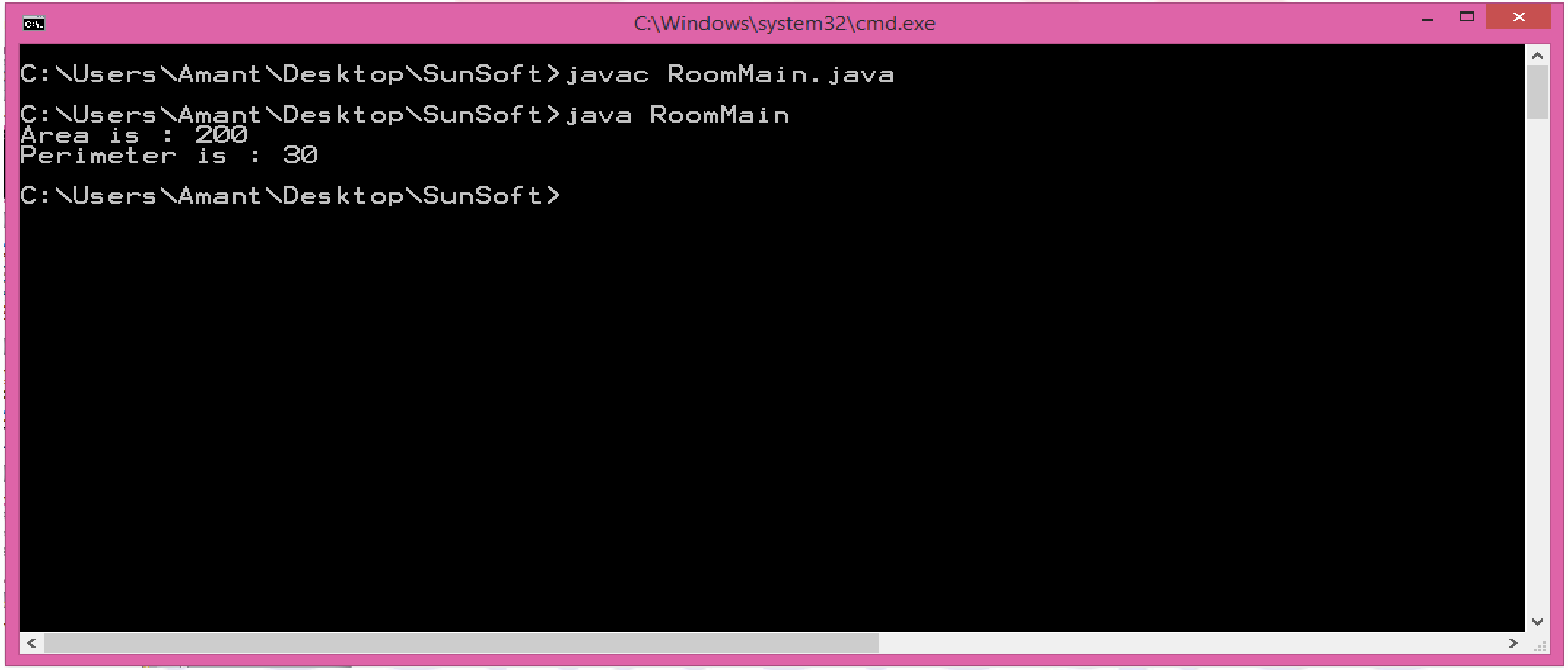
- We can’t override a constructor where method overriding is possible.

Constructors and overloading constructors (contd.)



```
class Room {
		public Room(int length, int width) {
			int area = length*width;
			System.out.println("Area is : "+area);
		}
		public Room(int length) {
			int width = 10;
			int peri = length+width;
			System.out.println("Perimeter is : "+peri);
		}
}
public class RoomMain {
		public static void main(String ag[]) {
			Room r1 = new Room(20,10);
			Room r2 = new Room(20);
		}
}
```

Constructors and overloading constructors (contd.)



```
C:\Users\Amant\Desktop\SunSoft>javac RoomMain.java

C:\Users\Amant\Desktop\SunSoft>java RoomMain
Area is : 200
Perimeter is : 30

C:\Users\Amant\Desktop\SunSoft>
```

instance and static variables

instance variables:

- Instance variables are declared inside a class but outside any block like methods or constructors.
- Instance variables are created when an object is created and destroys when the object is destroyed.
- Instance variables are stored in heap memory.
- The scope of instance variables is throughout the class.
- Instance variables can be directly invoked by calling the variable name.
- Default values of instance variables:

• int	‘0’
• boolean	‘false’
• String	‘null’

instance and static variables (contd.)

static variables:

- ➢ Static variables are also called as ‘Class variables’.
- ➢ Static variables are declared inside a class with ‘static’ keyword, but outside any block like methods or constructor.
- ➢ Static variables have only one copy throughout the class regardless of number of object created.
- ➢ Static variables are stored in static memory.
- ➢ It is mostly used to declare constants.
- ➢ Default values of static variables are same as instance variables.
- ➢ Static variables can be invoked by calling the class name.
  - ClassName.variable_name

instance and static methods

instance methods:

- Instance methods are declared normally i.e. without static keyword.
- Instance methods belong to the object of the class.
- Instance methods can be called after creating the object of the class.
- Instance methods are stored in permanent generation of heap memory but their parameters and their local variables and value to be returned are stored in stack.
- Instance methods can be overridden.

instance and static methods (contd.)

static methods:

- Static methods belongs to class.
- Static methods are declared by using static keyword.
- Just like static variables, only one copy of static method is passed throughout the class regardless of number of object created.
- Static methods are stored in permanent generation of heap memory.
- Static methods can be overloaded but cannot be overridden.
- Static methods can be invoked by calling the class name.
  - ClassName.methodName()
- Static methods can’t access instance methods and instance variables directly but instance method can access static variables and static methods directly.

instance and static block

instance block:

- Instance block is executed when the instance of the class is created.
- Instance block executes after the execution of static block.
- ‘this’ keyword can be used with instance block.
- Both static and non-static variables can be accessed inside instance block.
- Instance blocks are basically used for initializing instance variables or calling any instance method.

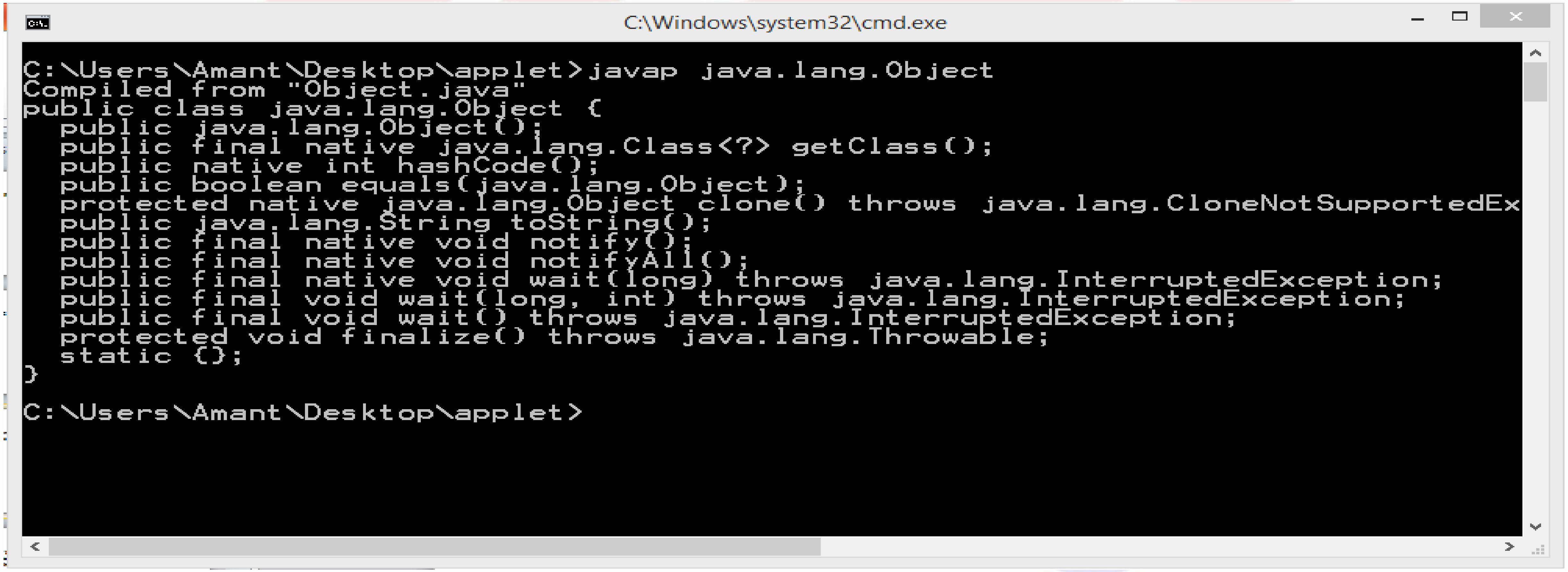
instance and static block (contd.)

static block:

- Static block is executed when the class is loaded.
- Static block executes before instance block.
- ‘this’ keyword cannot be used inside static block.
- Static blocks are basically used for initializing static variables or calling any static method.

Javap command

javap command is used to see the properties of a class. It displays information about the fields, constructors and methods present in a class file.

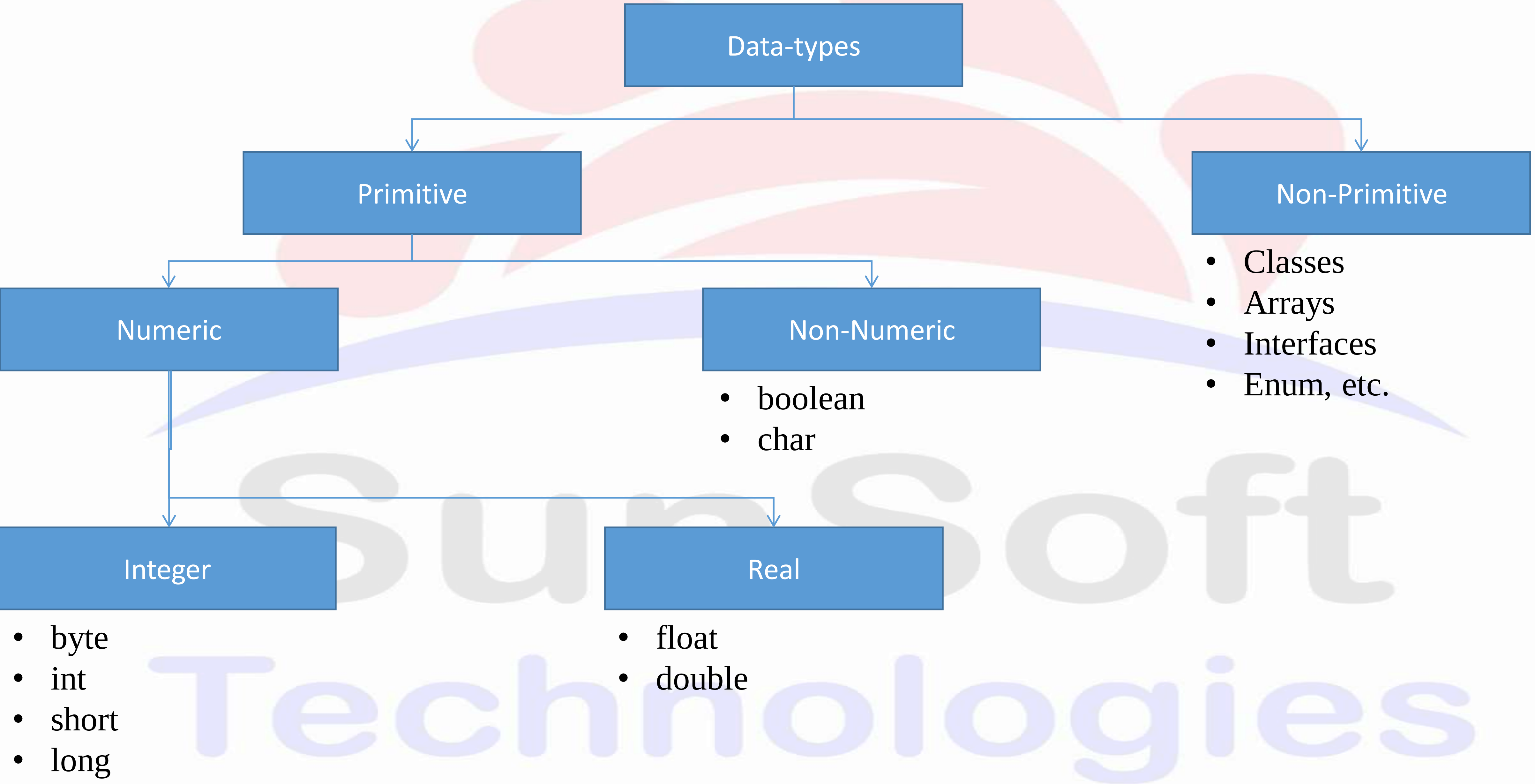


```
C:\Users\Amant\Desktop\applet>javap java.lang.Object
Compiled from "Object.java"
public class java.lang.Object {
  public java.lang.Object();
  public final native java.lang.Class<?> getClass();
  public native int hashCode();
  public boolean equals(java.lang.Object);
  protected native java.lang.Object clone() throws java.lang.CloneNotSupportedEx
  public java.lang.String toString();
  public final native void notify();
  public final native void notifyAll();
  public final native void wait(long) throws java.lang.InterruptedException;
  public final void wait(long, int) throws java.lang.InterruptedException;
  public final void wait() throws java.lang.InterruptedException;
  protected void finalize() throws java.lang.Throwable;
  static {};
}

C:\Users\Amant\Desktop\applet>
```

Data-types

Data-types are the keyword which defines what type of value a variable may hold.

- There is no ‘unsigned’ keyword in java.
- There is no ‘short int’ or ‘long int’ in java.


Data-types
Primitive
Non-Primitive
Classes
Arrays
Interfaces
Enum, etc.
Numeric
Non-Numeric
boolean
char
Integer
Real
byte
int
short
long
float
double

Literals

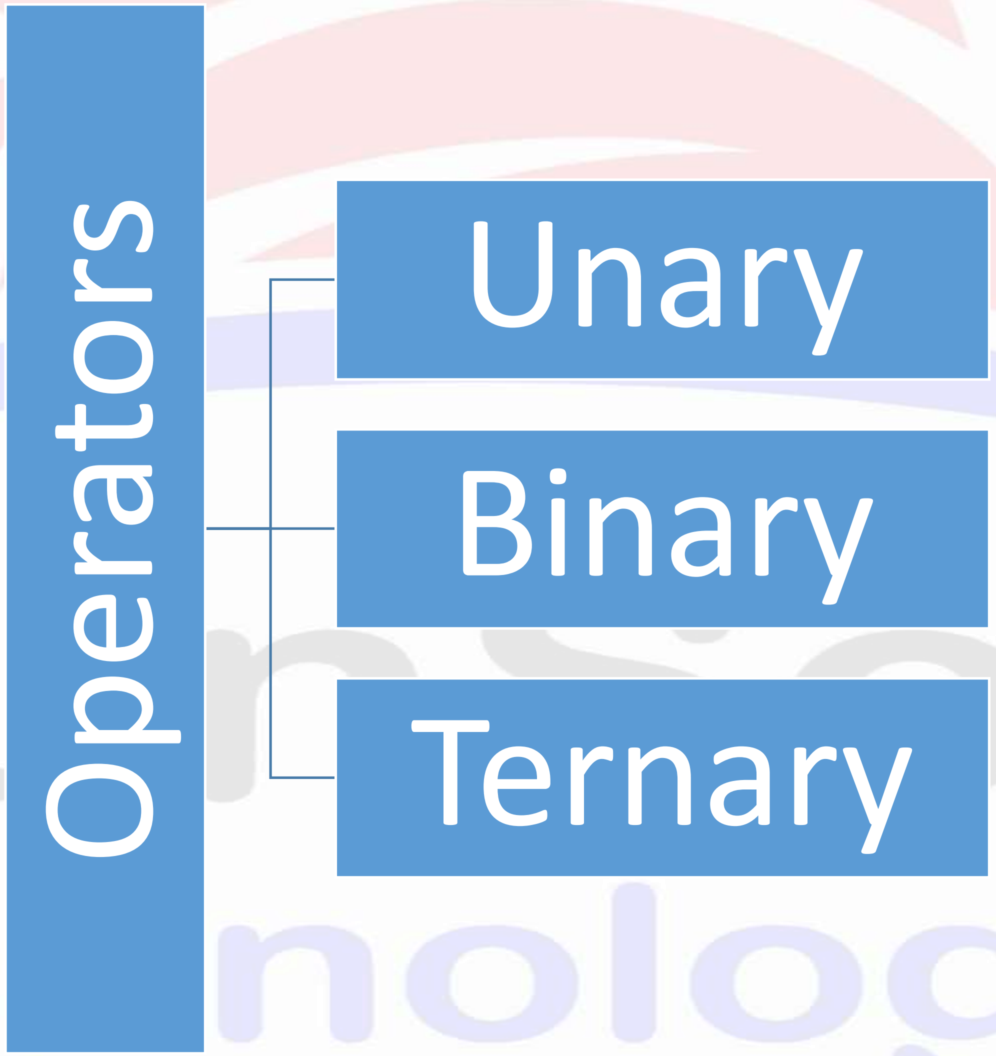
A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

- Literals can be assigned to any primitive type variable.
  - E.g.: int a = 10;
- Prefix ‘0’ is used to indicate octal, and prefix ‘0x’ indicates hexadecimal when using these number systems of literals.
  - E.g.: int decimal=10;
    int octal=014;
    int hexa=0x10;

Operators

- **Variables**: Variables are entities to hold some value.
- **Constants**: Constants are something that cannot be changed.
- **Identifier**: Identifiers are the name given to any variable, class or method.
  - '$', '_' are allowed and can be used in the beginning of any identifier's name.
- **Operators**: Operators are any special character which performs operations on operands.
- **Operands**: Operands are anything on which operator operates.

Types of Operators



Types of Operators (contd.)

Arithmetic Operators: + - * / %

Logical Operators: && || !

Relational/Comparison Operators: == != > < >= <==

Bitwise Operators: & | ^ !

Shift Operators: >> << >>>

Conditional/Ternary/Tertiary/Question-mark Operator: int res=(a>b)?a:b;

Assignment/Short-hand Operators: = += == *= /= %=

Dot Operator: .

Increment/Decrement Operators:

- Pre-increment ++
- Pre-decrement --
- Post-increment ++
- Post-decrement --

Conditionals statements

In Java we have four types of conditional statements:

1. if statement
2. if-else statement
3. if-else if statement
4. switch statement

Conditionals statements (contd.)

if statement:

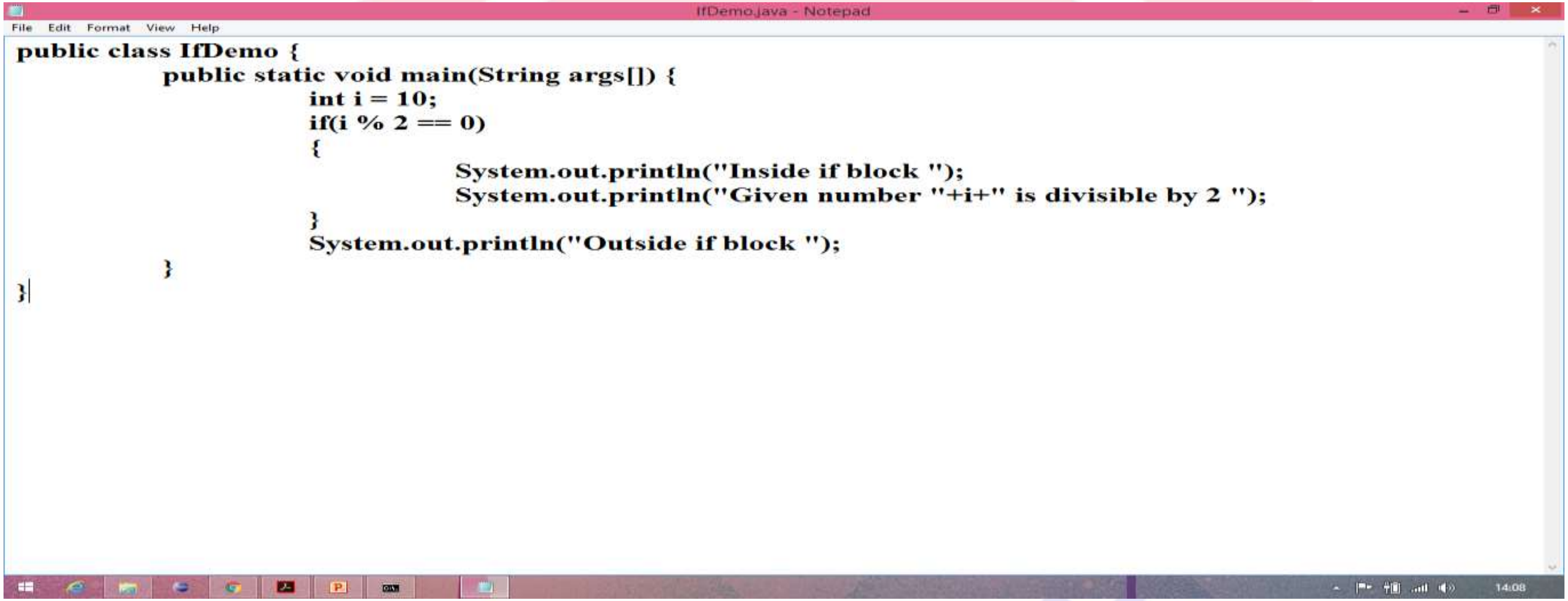
if statement is used if we want to execute a block of code if the condition is true.

Syntax:

```
if(condition)
{
        block to be executed if the condition is true
}
```

Conditionals statements (contd.)

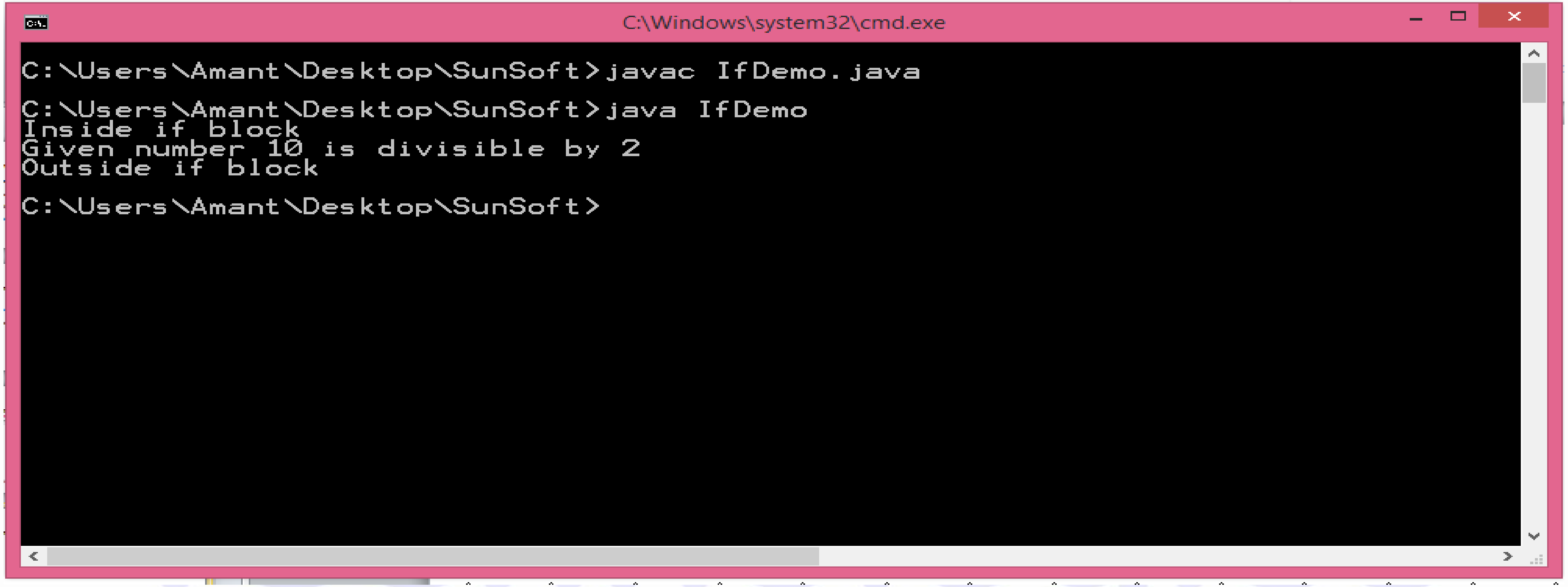
if statement demo:



```
public class IfDemo {
		public static void main(String args[]) {
			int i = 10;
			if(i % 2 == 0)
			{
				System.out.println("Inside if block ");
				System.out.println("Given number "+i+" is divisible by 2 ");
			}
			System.out.println("Outside if block ");
		}
}
```

Conditionals statements (contd.)

if statement demo:



```
C:\Users\Amant\Desktop\SunSoft>javac IfDemo.java

C:\Users\Amant\Desktop\SunSoft>java IfDemo
Inside if block
Given number 10 is divisible by 2
Outside if block

C:\Users\Amant\Desktop\SunSoft>
```

Conditionals statements (contd.)

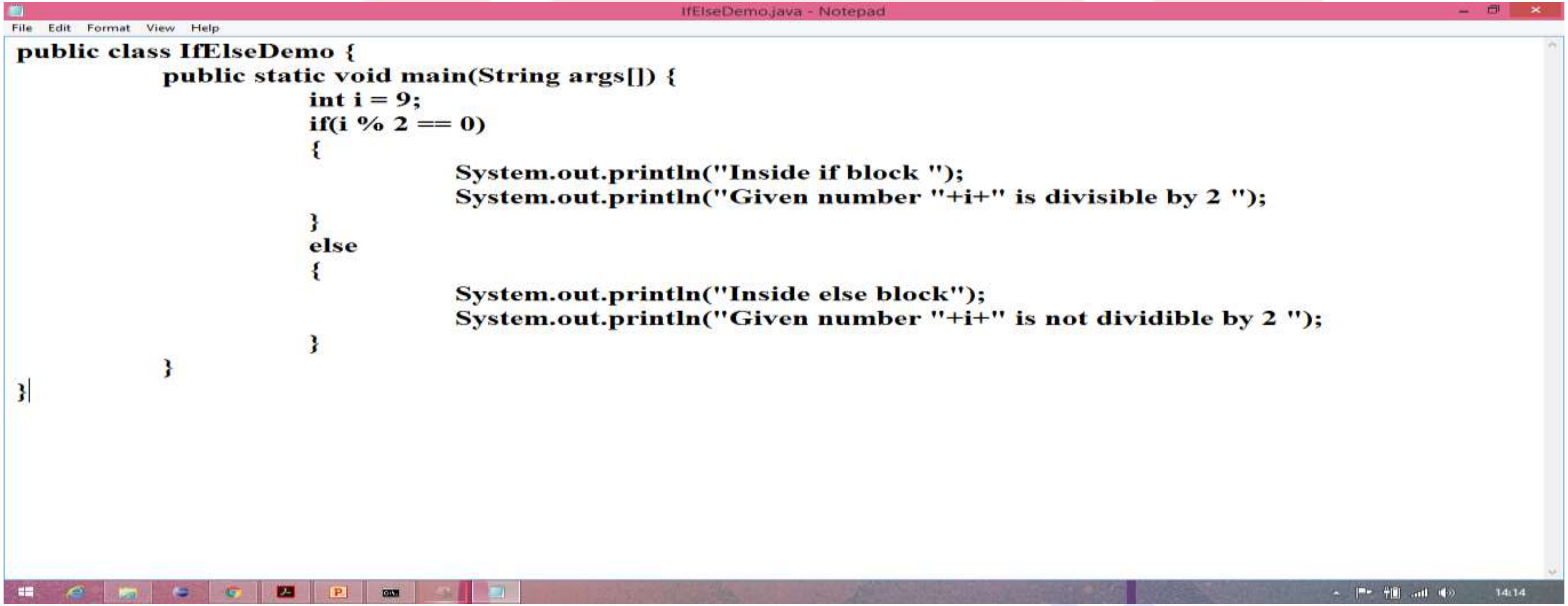
if else statement: if else statement is used if we want to execute a block of code if the condition is true otherwise else block is executed.

Syntax:

```
if(condition)
{
        block to be executed if the condition is true
}
else
{
        block to be executed if the condition is false
}
```

Conditionals statements (contd.)

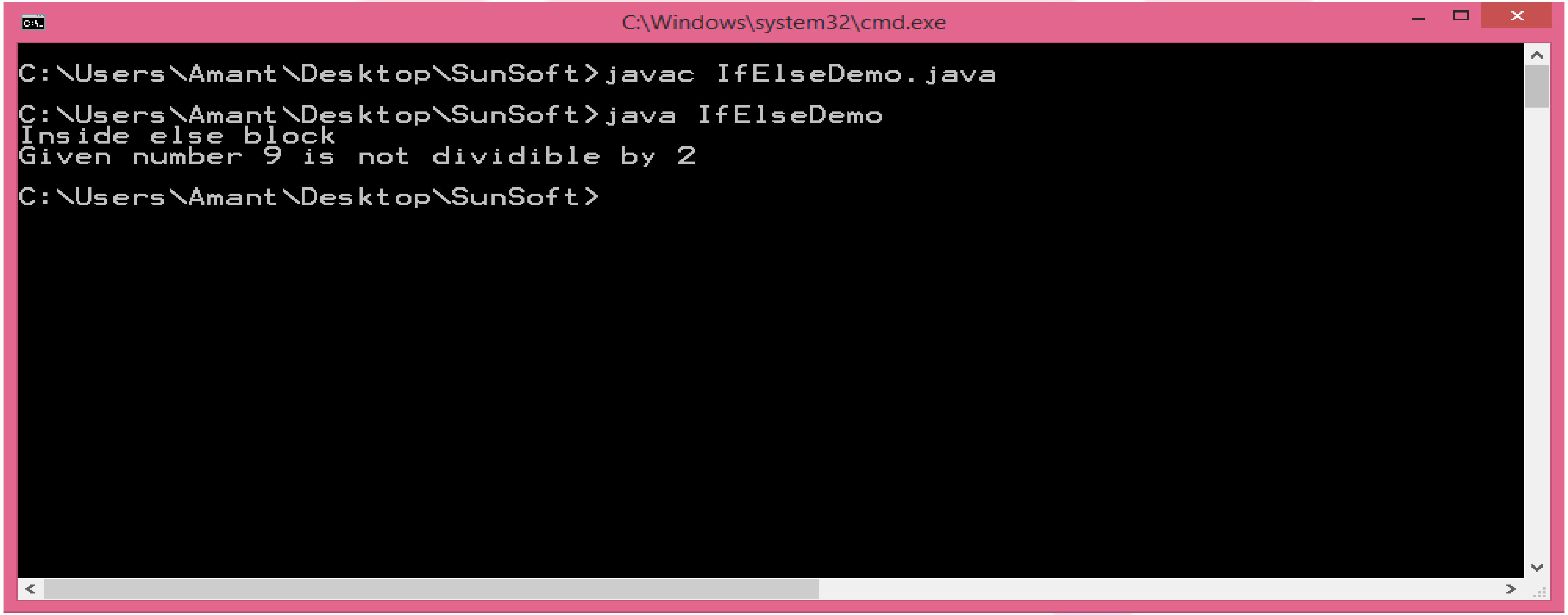
if else statement demo:



```
public class IfElseDemo {
		public static void main(String args[]) {
				int i = 9;
				if(i % 2 == 0)
				{
						System.out.println("Inside if block ");
						System.out.println("Given number "+i+" is divisible by 2 ");
				}
				else
				{
						System.out.println("Inside else block");
						System.out.println("Given number "+i+" is not dividible by 2 ");
				}
		}
}
```

Conditionals statements (contd.)

if else statement demo:



```
C:\Users\Amant\Desktop\SunSoft>javac IfElseDemo.java

C:\Users\Amant\Desktop\SunSoft>java IfElseDemo
Inside else block
Given number 9 is not dividible by 2

C:\Users\Amant\Desktop\SunSoft>
```

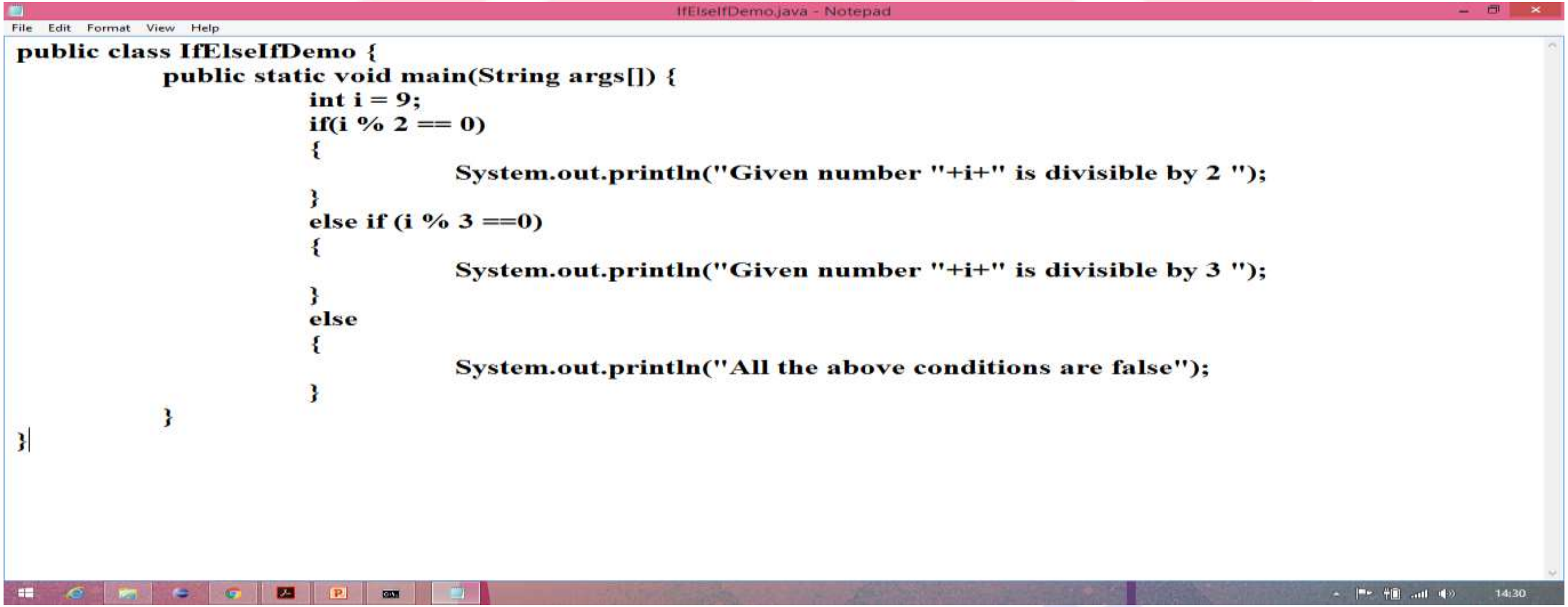
Conditionals statements (contd.)

if-else if statement: if-else if can be used when we have to check multiple conditions. **Syntax:**

```
if(condition)
{
        block to be executed if this condition is true
}
else if(condition)
{
        block to be executed if this condition is true
}
else {
        block to be executed if all the conditions are false
}
```

Conditionals statements (contd.)

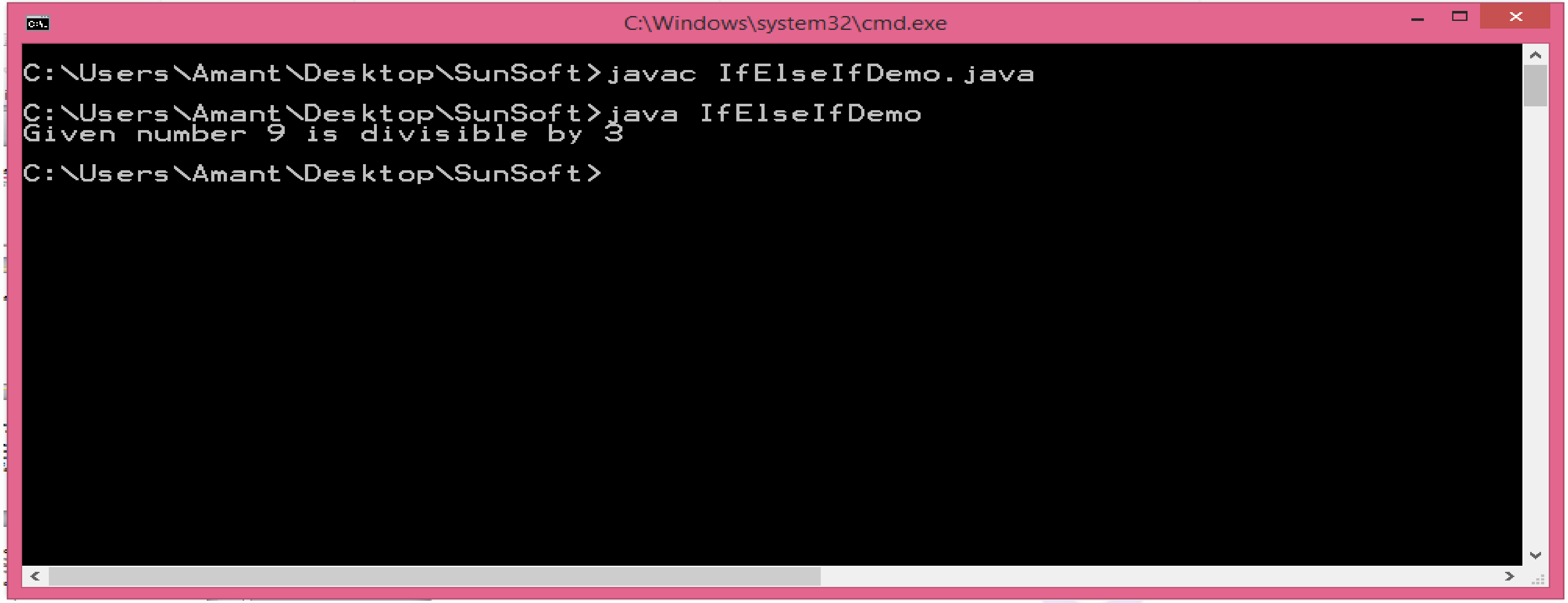
if-else if statement demo:



```
public class IfElseIfDemo {
		public static void main(String args[]) {
				int i = 9;
				if(i % 2 == 0)
				{
						System.out.println("Given number "+i+" is divisible by 2 ");
				}
				else if (i % 3 ==0)
				{
						System.out.println("Given number "+i+" is divisible by 3 ");
				}
				else
				{
						System.out.println("All the above conditions are false");
				}
		}
}
```

Conditionals statements (contd.)

if-else if statement demo:



```
C:\Users\Amant\Desktop\SunSoft>javac IfElseIfDemo.java

C:\Users\Amant\Desktop\SunSoft>java IfElseIfDemo
Given number 9 is divisible by 3

C:\Users\Amant\Desktop\SunSoft>
```

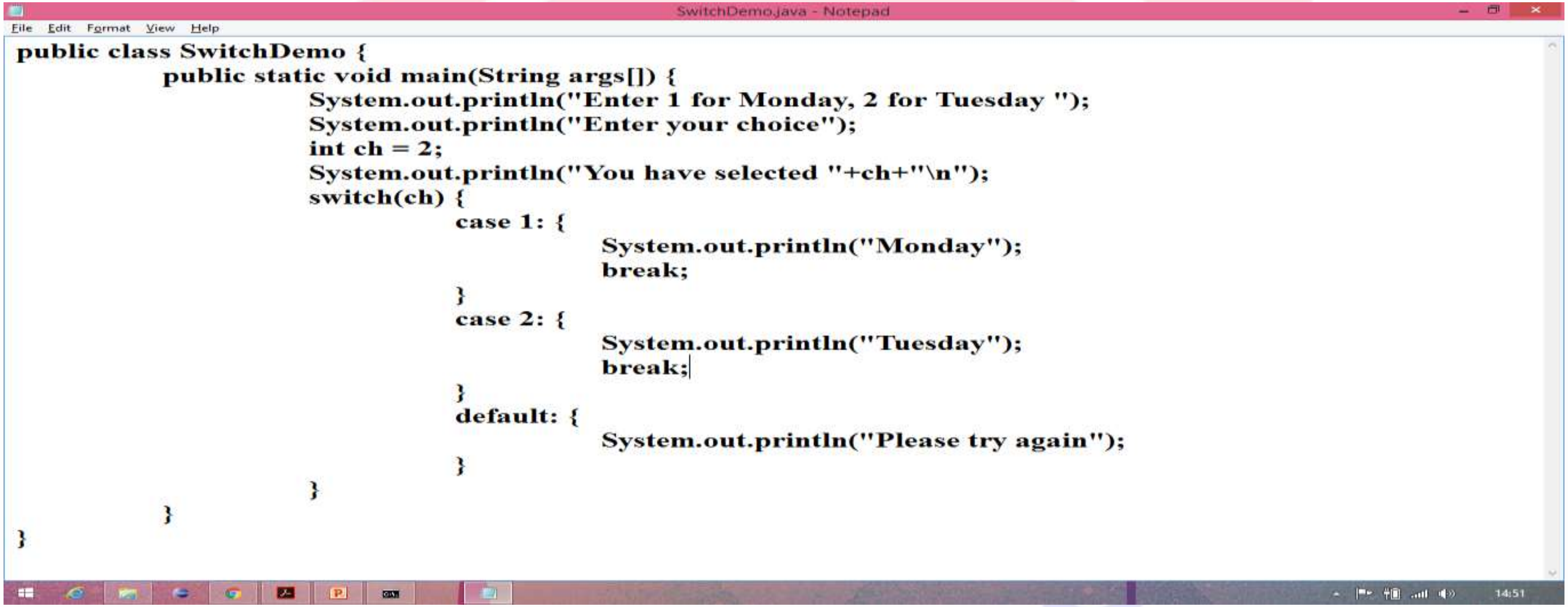
Conditionals statements (contd.)

switch statement: switch statements can be used when we have to check multiple conditions and execute any one according to user's choice.

Syntax:
```
switch(choice) {
          case 1:{
                    statements
               }
          case 2:{
                    statements
               }
          default :{
                    statements
               }}
```

Conditionals statements (contd.)

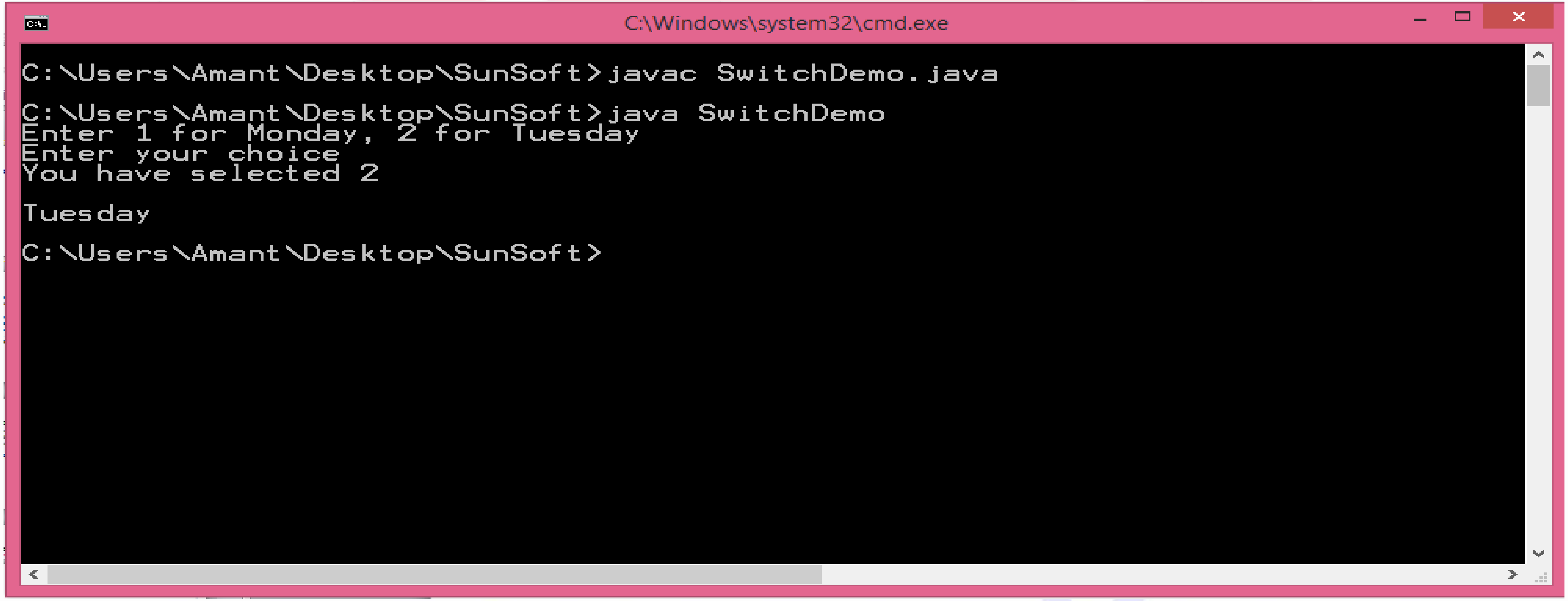
switch statement demo:



```
public class SwitchDemo {
		public static void main(String args[]) {
			System.out.println("Enter 1 for Monday, 2 for Tuesday ");
			System.out.println("Enter your choice");
			int ch = 2;
			System.out.println("You have selected "+ch+"\n");
			switch(ch) {
				case 1: {
					System.out.println("Monday");
					break;
				}
				case 2: {
					System.out.println("Tuesday");
					break;
				}
				default: {
					System.out.println("Please try again");
				}
			}
		}
}
```

Conditionals statements (contd.)

switch statement demo:



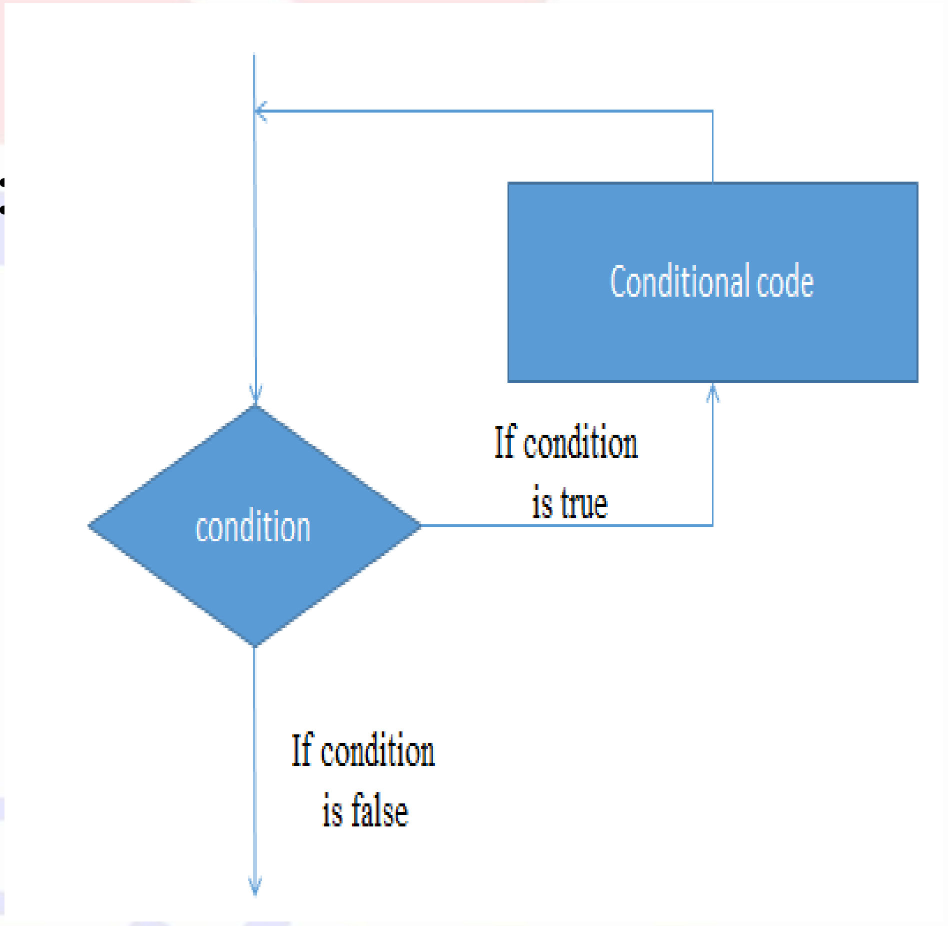
```
C:\Users\Amant\Desktop\SunSoft>javac SwitchDemo.java

C:\Users\Amant\Desktop\SunSoft>java SwitchDemo
Enter 1 for Monday, 2 for Tuesday
Enter your choice
You have selected 2

Tuesday

C:\Users\Amant\Desktop\SunSoft>
```

Loops

Looping statements are used to execute a block of code more than one time depending upon the scenario.

In java there are four types of looping statements:

1. while loop
2. do while loop
3. for loop
4. nested for loop



Loops (contd.)

while loop:

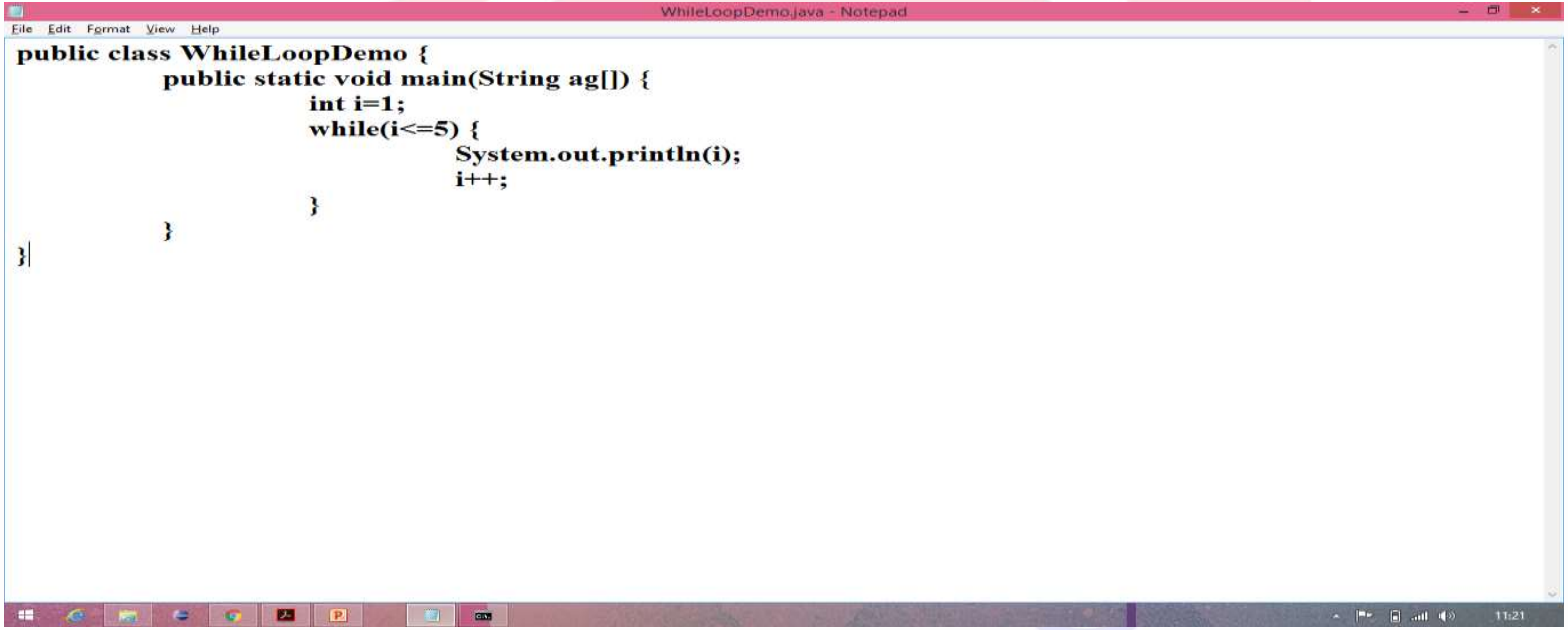
- while loop is used to iterate a part of the program several times.
- If the number of iteration is not fixed, it is recommended to use while loop.

- Syntax:

```
while (condition)
{
        block of code to be executed
        expression
}
```

Loops (contd.)

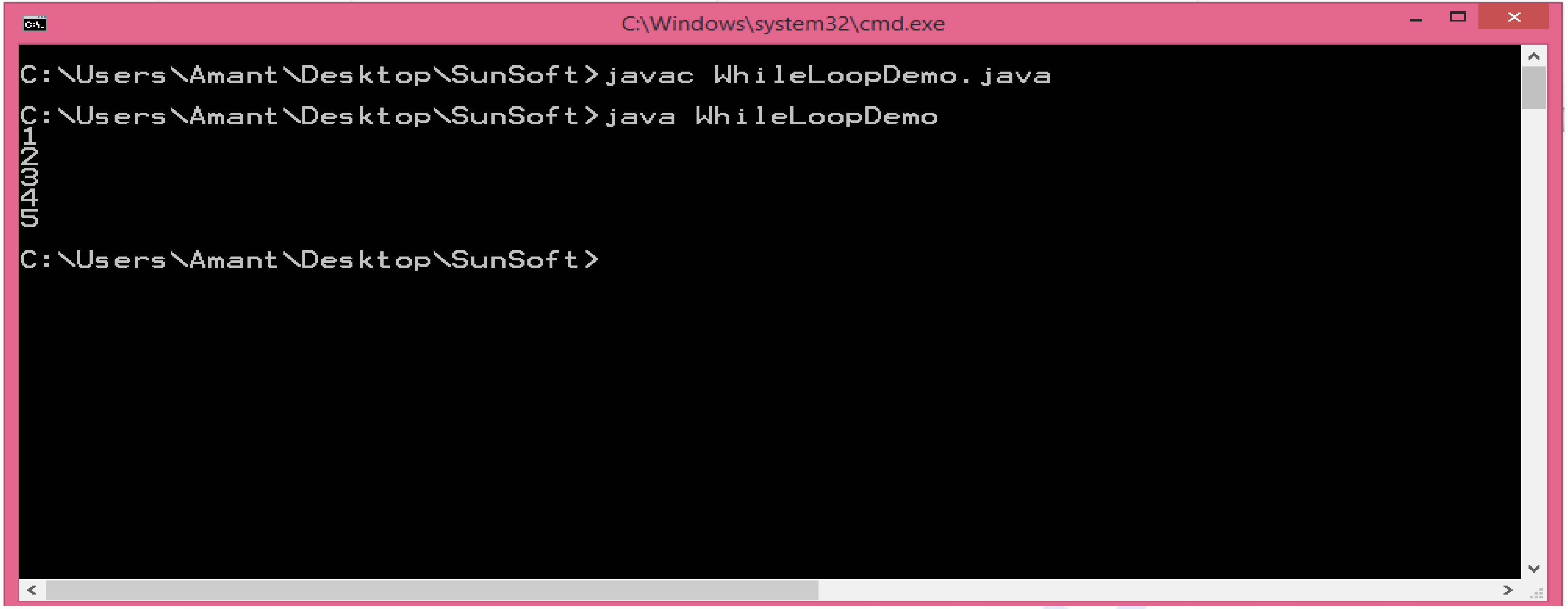
while loop example:



```
public class WhileLoopDemo {
		public static void main(String ag[]) {
			int i=1;
			while(i<=5) {
				System.out.println(i);
				i++;
			}
		}
}
```

Loops (contd.)

while loop example:



```
C:\Users\Amant\Desktop\SunSoft>javac WhileLoopDemo.java

C:\Users\Amant\Desktop\SunSoft>java WhileLoopDemo
1
2
3
4
5

C:\Users\Amant\Desktop\SunSoft>
```

Loops (contd.)

do while loop:

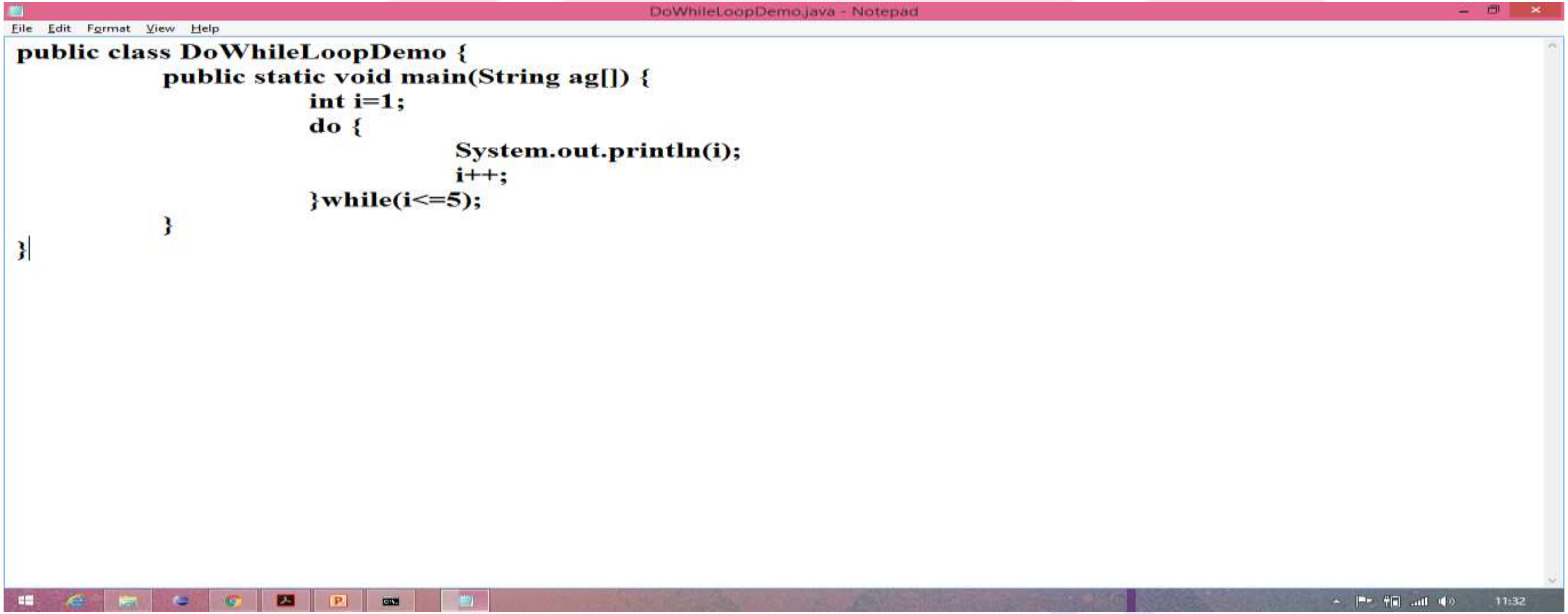
- do while loop is used to iterate a part of the program several times.
- If the number of iteration is not fixed and you must have to execute the loop at least once then it is recommended to use do while loop.

- Syntax:

```
do {
        block of code to be executed
        expression
} while (condition);
```

Loops (contd.)

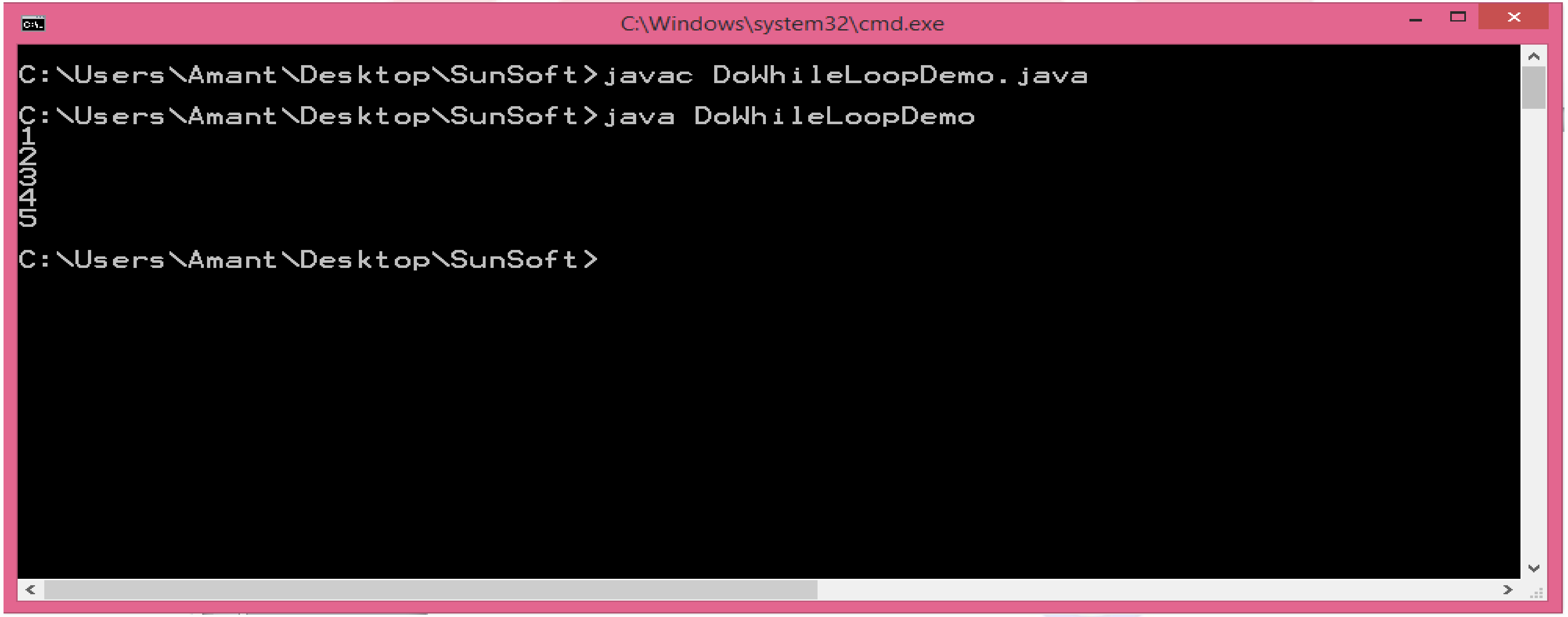
do while loop example:



```
public class DoWhileLoopDemo {
		public static void main(String ag[]) {
				int i=1;
				do {
						System.out.println(i);
						i++;
				}while(i<=5);
		}
}
```

Loops (contd.)

do while loop example:



C:\Windows\system32\cmd.exe

```
C:\Users\Amant\Desktop\SunSoft>javac DoWhileLoopDemo.java

C:\Users\Amant\Desktop\SunSoft>java DoWhileLoopDemo
1
2
3
4
5

C:\Users\Amant\Desktop\SunSoft>
```

Loops (contd.)

for loop:

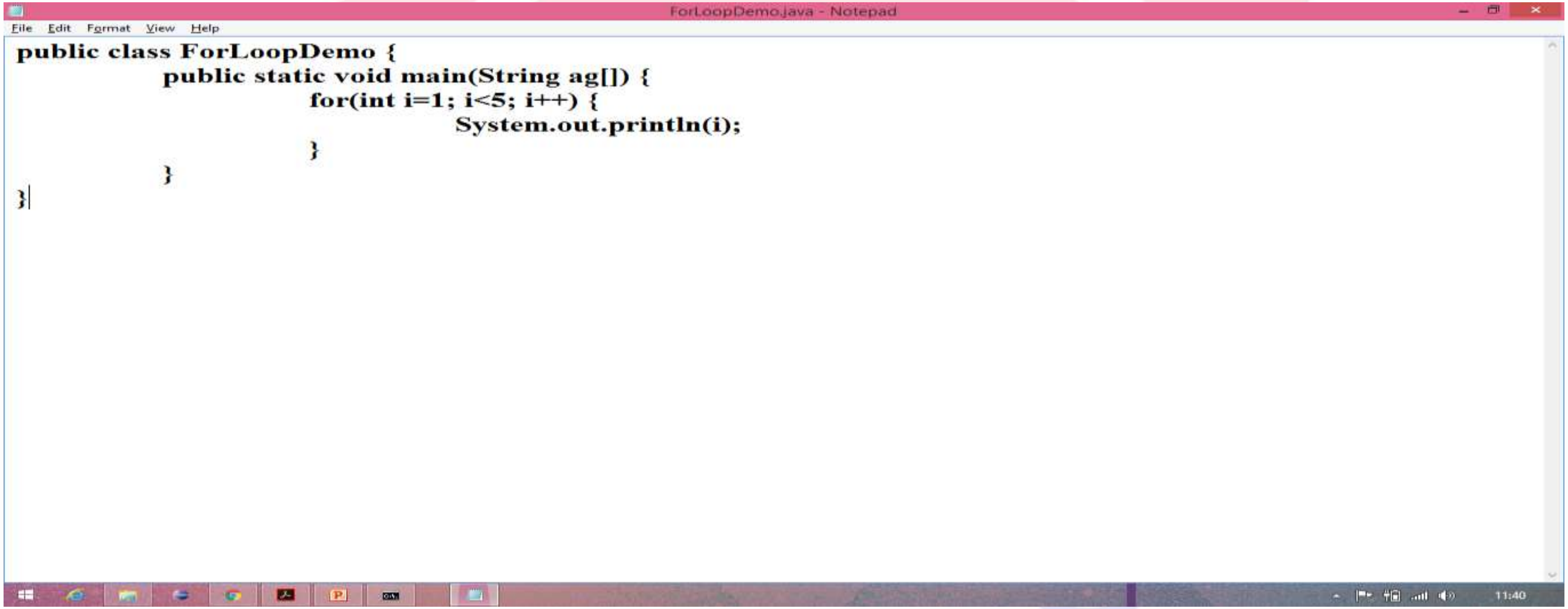
- for loop is used to iterate a part of the program several times.
- If the number of iteration is fixed then it is recommended to use do while loop.

- Syntax:

```
for(initialization; condition; expression) {
        block of code to be executed
}
```

Loops (contd.)

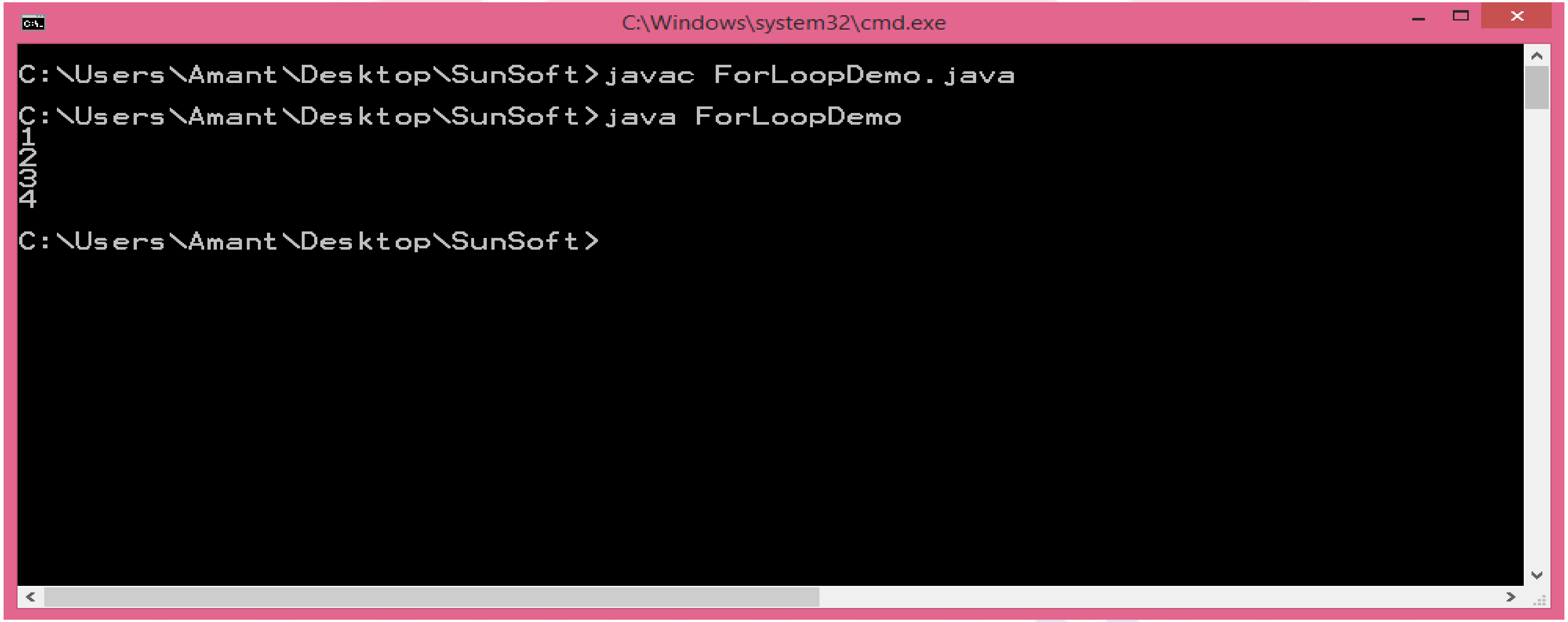
for loop example:



```
public class ForLoopDemo {
        public static void main(String ag[]) {
                for(int i=1; i<5; i++) {
                        System.out.println(i);
                }
        }
}
```

Loops (contd.)

for loop example:



```
C:\Users\Amant\Desktop\SunSoft>javac ForLoopDemo.java

C:\Users\Amant\Desktop\SunSoft>java ForLoopDemo
1
2
3
4

C:\Users\Amant\Desktop\SunSoft>
```

Enhanced for loop

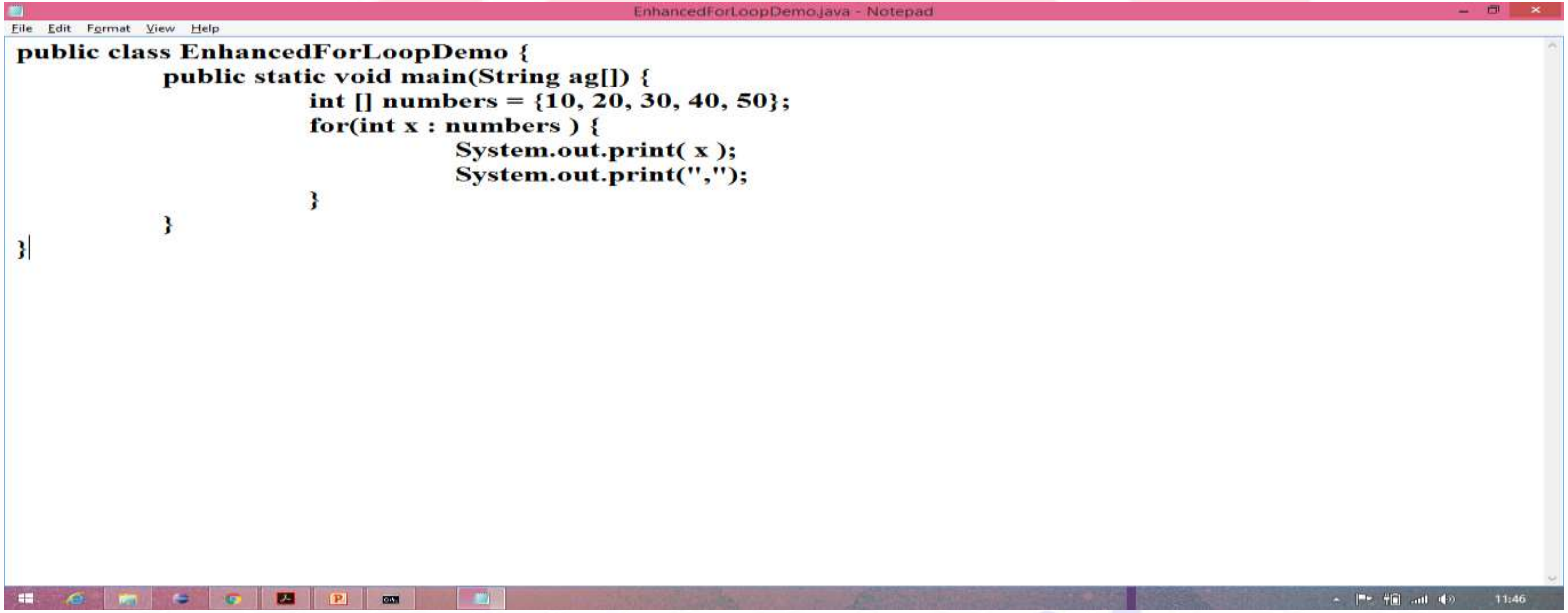
- Enhanced for loop is used to traverse collection of elements including arrays.
- Enhanced for loop was introduced in Java 5.

Syntax:

```
for(declaration : expression) {

        block of code to be executed

}
```

Enhanced for loop (contd.)

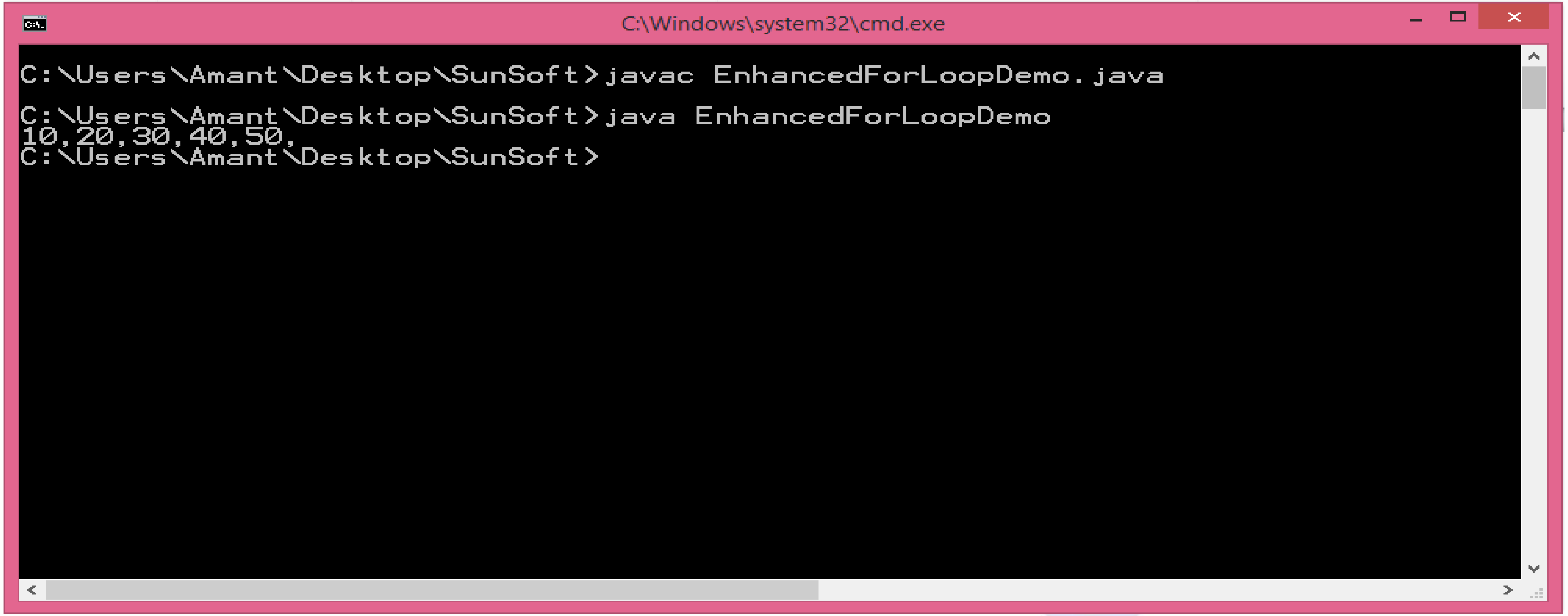
Enhanced for loop example:


EnhancedForLoopDemo.java - Notepad
File Edit Format View Help

```
public class EnhancedForLoopDemo {
		public static void main(String ag[]) {
				int [] numbers = {10, 20, 30, 40, 50};
				for(int x : numbers ) {
						System.out.print( x );
						System.out.print(",");
				}
		}
}
```

Enhanced for loop (contd.)

Enhanced for loop example:



```
C:\Users\Amant\Desktop\SunSoft>javac EnhancedForLoopDemo.java

C:\Users\Amant\Desktop\SunSoft>java EnhancedForLoopDemo
10,20,30,40,50,
C:\Users\Amant\Desktop\SunSoft>
```

Command line arguments

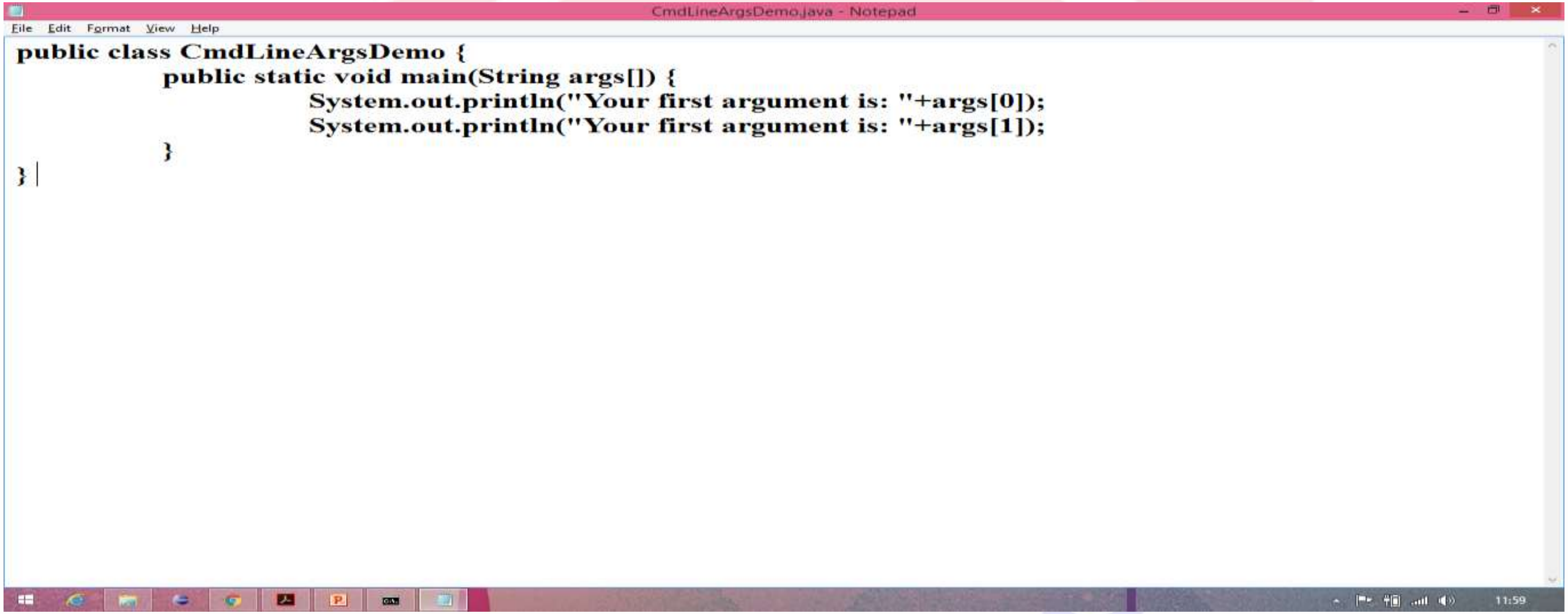
- The command line argument is the argument passed to a program at the time when you run it.

- To access the command-line argument inside a java program is quite easy, they are stored as string in String array passed to the args parameter of main() method.

public static void main(String args[])

Command line arguments (contd.)

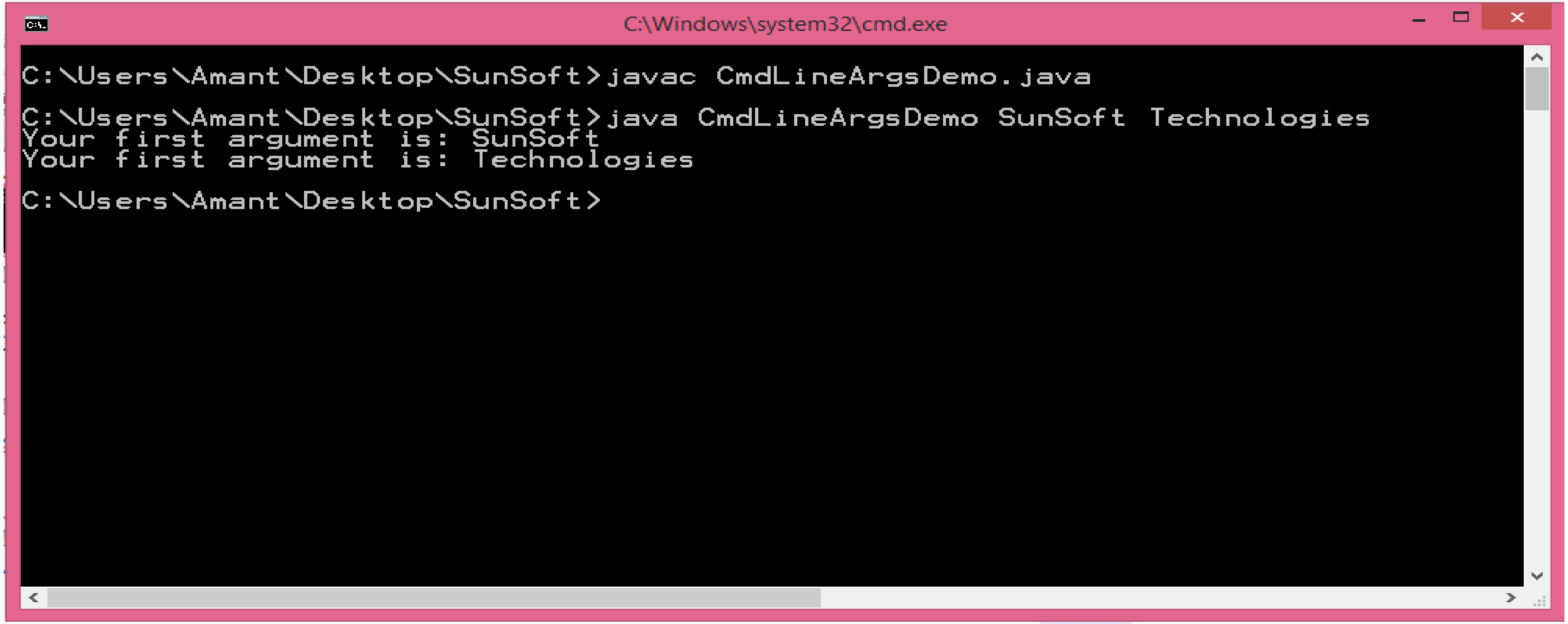
Command line arguments example:



```
public class CmdLineArgsDemo {
		public static void main(String args[]) {
			System.out.println("Your first argument is: "+args[0]);
			System.out.println("Your first argument is: "+args[1]);
		}
}
```

Command line arguments (contd.)

Command line arguments example:



C:\Windows\system32\cmd.exe

```
C:\Users\Amant\Desktop\SunSoft>javac CmdLineArgsDemo.java

C:\Users\Amant\Desktop\SunSoft>java CmdLineArgsDemo SunSoft Technologies
Your first argument is: SunSoft
Your first argument is: Technologies

C:\Users\Amant\Desktop\SunSoft>
```

Wrapper classes

- Wrapper class was introduced in Java 5.

Wrapper classes	Primitive types
Byte	byte
Integer	int
Short	short
Long	long
Float	float
Double	double
Character	char
Boolean	boolean

Wrapper classes (contd.)

Unboxing:

- The phenomenon of converting object into primitive is called **unboxing.**
  - **E.g.:**

```
Integer n=new Integer(5);
int n1=n.intValue();                    //converting Integer to int (unboxing)
```

Autoboxing:

- The phenomenon of converting primitive into object is called **autoboxing.**
  - **E.g.:**

```
int n=20;
Integer n1=Integer.valueOf(n);//converting int into Integer (autoboxing)
```

Keyword ‘this’

- ‘this’ is a keyword which refer to the same class or current class.
- While calling constructor using this key word ,this keyword should be the first statement.
- While calling variable and methods it can be any where.

Example for this key word

Demo :

```
class A
          {
                    int a=34;
                    void add(){}
          }
          class B extends A
          {
                    int a=22;                    System.out.println(this.a);
          }
```

Difference between constructors and methods

constructors

- Constructors must have the same name as the class name.
- Non access modifiers can’t be applied with constructors.
- Constructors are automatically called when the object of the class is created.
- Constructors can be called only once for one object.
- Constructors don’t have any return type.
- Constructs can’t be overridden.

methods

- Methods can be have any name including the class name.
- Non access modifiers can be applied with methods.
- Methods are needed to be called explicitly.
- Methods can be called any number of times.
- Methods have return type.
- Methods can be overridden.

OOPS CONCEPT

- There are 2 famous programming models

  1. pops

  2.oops

1.Procedure oriented programming model

- The main building blocks of this model is function
- Any function can access any data
- Programming is more complex

2.Object oriented programming model

- The main building blocks of this model is Object
- Programming is more simple
- Concept of encapsulation and data security code reusability
- adding more function is not difficult
- Objects are independent of each other

Difference between oops and pops

oops	pops
• Object are main building blocks	• Function are main building block
• Modification is easy because Object are independent of each other	• Modification is difficult because functions are dependent on each other
• It provide access modifier so more secure	• It is not provide any access modifier so less secure
• Ex: c language	• Ex : java and c++

Principle of oops

- ➢There are total six oops concept

1. Encapsulation
2. Inheritance
3. Polymorphism
4. State and behaviour should be declare inside the class
5. Every thing should be in terms of classes and Objects
6. Every operation should be done by invoking method

- ➢Java is not 100% Object Oriented Programing language
- ➢Java supports only first 4 features of oops so Java is pure Object oriented Programming language
- ➢Small talk support all feature of oops so it is 100% oops
- ➢C++ support first 3 feature of oops so it is object oriented programming language
- ➢C does not support oops

1. Encapsulation

- ➢In Encapsulation the data and function are declare inside class itself.
- ➢Abstraction is concept of data hiding and is part of encapsulation

ex :class, capsule

```
        class {
                int a=10;
                void add(){
                }
        }
```

a . private fields

➢We can create pure Encapsulation class by declaring all data as private

setter method

➢Is used to access and set the value of private data in other class

getter method

➢It is used to allow other class to access the private data

Differences between java bean class and normal class

java bean class	Normal class
java bean class must be declare as public	Normal class can be public or default
All instance variable must be declare as private	variable can have any access modifier
getter and setter is used to access the data	Depending upon access modifier we can access the data
It should have default constructor only	Any constructor is allowed

Inheritance

- It is used to inherit the property of an old class to a new class
- ex:

```
class A{
        int i=10;
        }
class B extends A{
        System.out.println(i);
        }
```

- We can inherit data form super class to subclass but revers is not possible

Types of inheritance

1.Single Inheritance

2.Multilevel Inheritance

3.Multiple inheritance

4.Hierarical Inheritance

1.single Inheritance

➢This is simple one inheriting property from one class to other class

Demo :

```
class A{
            int i=10;
            }
      class B extends A{
            System.out.println(i);
            }
```

2.Multilevel Inheritance

- One super class for each sub class

Demo :

```
class A{
            int i=10;
            }
      class B extends A{
            System.out.println(i);
            }
      class C extends B{}
```

3.Multiple Inheritance

- Multiple super class for each single subclass

Demo :

```
class A{
          int i=10;
          }
     class B extends A{
          System.out.println(i);
          }
     class C extends A ,B{}
```

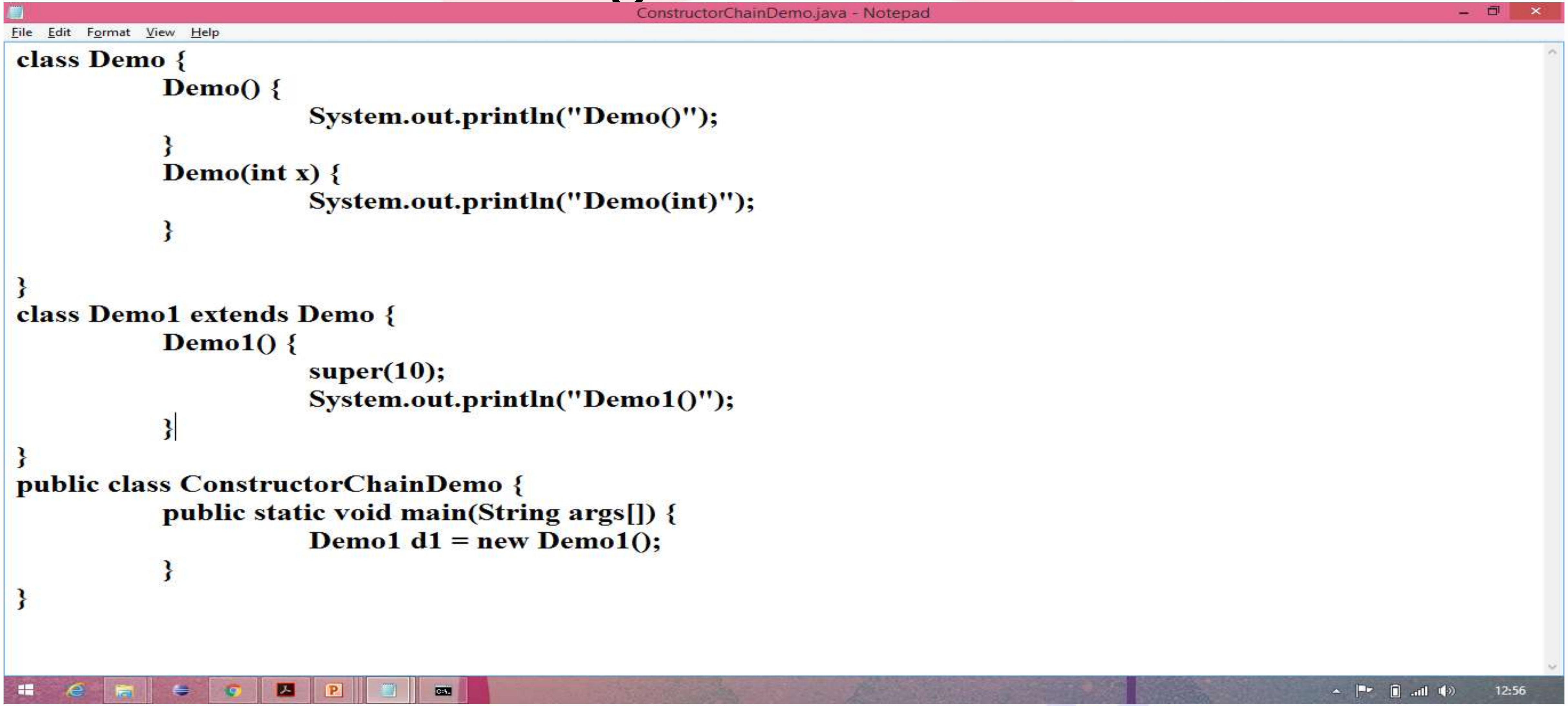
4.Hierarchal Inheritance

➢Multiple subclass for super class

Demo :

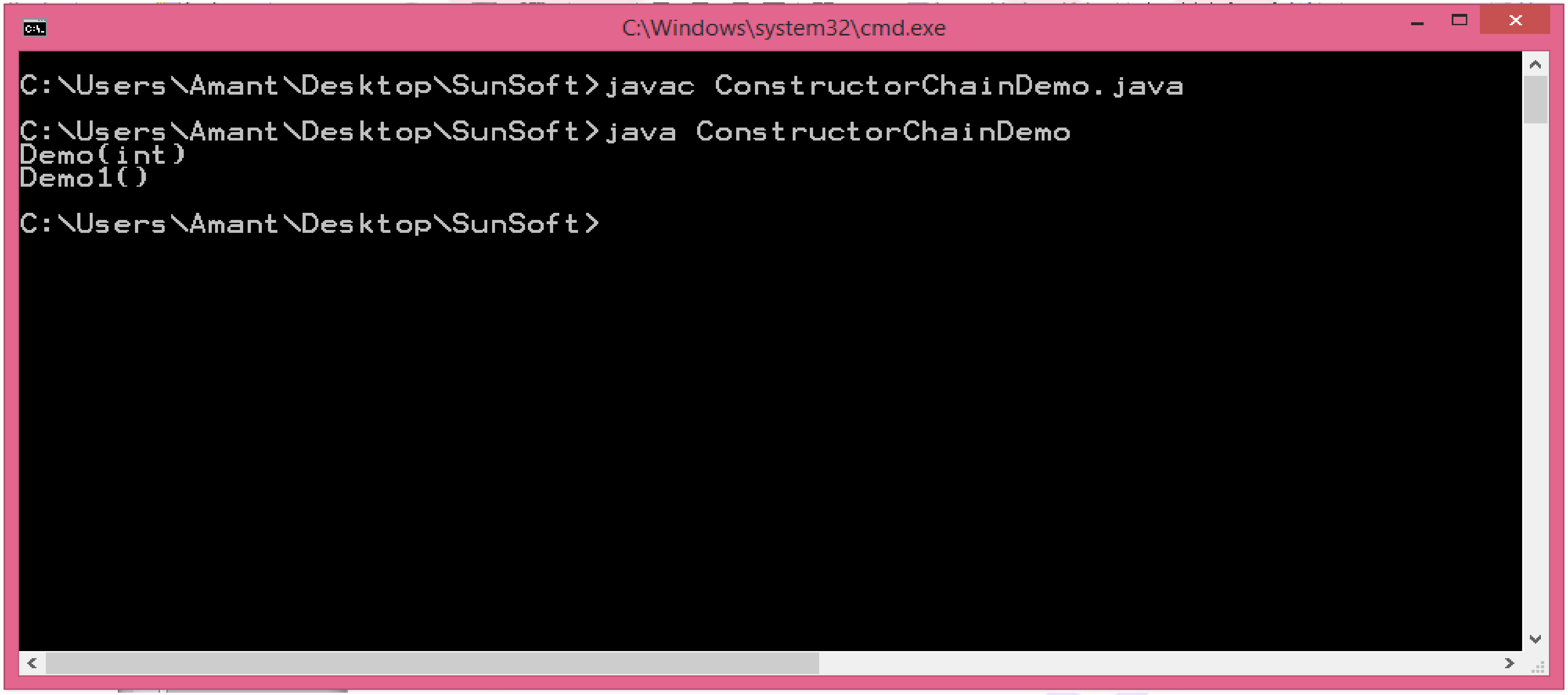
```
class A{
                int i=10;
        }
class B extends A
        {
                System.out.println(i);
        }
class C extends A {}
```

Constructor chaining in inheritance:


ConstructorChainDemo.java - Notepad
File Edit Format View Help
12:56

```
class Demo {
        Demo() {
                System.out.println("Demo()");
        }
        Demo(int x) {
                System.out.println("Demo(int)");
        }

}
class Demo1 extends Demo {
        Demo1() {
                super(10);
                System.out.println("Demo1()");
        }
}
public class ConstructorChainDemo {
        public static void main(String args[]) {
                Demo1 d1 = new Demo1();
        }
}
```

Constructor chaining in inheritance (contd.):



```
C:\Users\Amant\Desktop\SunSoft>javac ConstructorChainDemo.java

C:\Users\Amant\Desktop\SunSoft>java ConstructorChainDemo
Demo(int)
Demo1()

C:\Users\Amant\Desktop\SunSoft>
```

super

- Super is keyword used to refer super class member from subclass but not in reverse

- By default every subclass contain super class constructor which non-parameterized constructor

Example for super key word

Demo :

```
class A
        {
                int a=34;
                void add(){}
        }
class B extends A
        {
                super.add();
                System.out.println(super.a);
        }
```

Polymorphism

- In polymorphism Object will behave differently in deferent situation

Ex : press switch one time fan will on,

press it again fan will be off

- There are two types of polymorphism

1.compile time polymorphism

2.run time polymorphism

Run time polymorphism

- **Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- This can be achieved by method overriding

Method Overriding

- Overriding is the process of writing same method name with same signature in subclass

Demo :

```
class A
{
        void add(){}
}
class B extends A
{
         void add(){}
}
```

Compile time polymorphism

- Compile time polymorphism is a process in which a call to an overridden method is resolved at compile-time.
- This can be achieved by using methods overloading

Method Overloading

- Overloading is the process of writing same method name with different parameter in same class

Demo ;

```
class A{
        void add(){}
        void add(int a){}
        }
```

Abstraction

- Abstraction is used to hiding internal Implementation and showing required feature
- Abstract is key word applicable for methods and class
- Object can not be create for abstract class
- Method body should not be given for abstract method
- With abstract 100% abstraction is not possible

Abstraction Demo

```
        class A{
                abstract void add();
                }
        class B extends A{
                void add(){}
                }
        public class M{
                public static void main(String[] args){
        //A a=new A();
        B b=new B();
                }
}
```

interface

- With interface 100% abstraction can be achieved
- With Interface we can achieve multiple inheritance
- All variable in interface are default public static final
- All method are public abstract
- You can not create Object of Interface

Demo for Interface

```
interface I{
        public static final Int i=10;
                public abstract void add();
                }
class A implements I{
        public void add(){}
        }
```

- interface can not have constructer
- interface can extends another interface
- class can implements interface

Marker Interface

- The interface which does not contain any thing or empty
- it is just instruction to java

ex : cloneable interface , serializable interface

Non access modifier static abstract final

1. **static**

- static is key word applicable for variable method
- static member will load in memory when class is loading in to the memory
- static member can be access with class name without creating an Object of the class
- instance variable can not be declare in static member

Static variable

- static variables are also called as ‘Class variables’.
- static variables are declared inside a class with ‘static’ keyword, but outside any block like methods or constructor.
- static variables have only one copy throughout the class regardless of number of object created.
- static variables are stored in static memory.
- It is mostly used to declare constants.
- Default values of static variables are same as instance variables.
- static variables can be invoked by calling the class name.
  - className.variable_name

abstract

- abstract is a key word applicable to class method
- When we define method as abstract we should not declare body for that method
- When we declare class as abstract then we can not create Object for that class
- abstract class will have constructor

Keyword ‘final’

- ‘final’ is key word applicable for class and variable and method
- final variable value can not change it is constant
- final method can not be override
- final class cannot be extend

abstract classes

- The class which contain abstract key word is called abstract class
- abstract method will not have body
- if we declare any method as abstract then that class should be declare as abstract
- We can not create Object of abstract class
- abstract class will have constructor

- We have to implement abstract method in subclass of abstract class otherwise we need to declare that subclass as abstract class

- We can access all member of abstract by creating Object of super class

Demo :

```
abstract class A{
            int a=12;
            void show();
}
class B extends A{
            void show(){
                        System.out.println(“show”);
                        }
}
public class{
            public static void main(String[] args){
            B b=new B();
            Syste.out.println(b.a);
            b.show();
}
```

Differences between abstract class and interface

abstract class	interface
➢This can have abstract as well as normal method	➢This will have only abstract method
➢variable can be any instance, static final	➢variable are static final only
➢Constructor are present in abstract classes	➢Constructor are absent in interface
➢abstract class does not support multiple inheritance	➢interface support multiple inheritance
➢100% abstraction is not possible	➢100% abstraction can be achieved

Type Conversions

- ➢Type conversion is the process of converting one type to another type
- ➢There are 2 type of type conversion

  1 . implicit conversion

  2 . explicit conversion
- ➢Implicit conversion is done by jvm internally
- ➢Explicit conversion has to done by developer

auto widening and explicit narrowing

- ➢Auto widening is the process of converting lower data types to higher data types
- ➢This will done by jvm internally

Ex : if we define byte b=12; and

int a=b;

and try to print ‘a’ then jvm will convert byte value to int value implicitly

Demo

```
class A{
public static void main(String[] args)
{
        byte b=19;
        int a=b;
        byte c=a;
        System.out.println(b);          //convert into int
        //System.out.println(c);        //compilation error
}
}
```

explicit narrowing

- The process of converting wide data type to narrow data type is called explicit narrowing
- This has to be done by developer manually
- if we assign value greater then data type capacity then we will get compilation error
- We need to convert that data type to higher data type

Demo :

```
class A{
public static void main(String[] args)
{
int a=129
byte b=a;
//System.out.println(b); //error
byte bb=(byte)b;
System.out.println(bb); //129
}
}
```

auto upcasting and explicit downcasting

- When you assign sub class Object to super class reference variable then it is called upcasting

- jvm will automatically convert that subclass Object to super class Object so this is called auto upcasting

```
Demo ;
class A{
int i=5;
}
class B extends A{
int f=45;
}
class Demo{
public static void main(String[] args){
B b=(B)new A();  //upcasting
}
```

explicit downcasting

- When you assign super class Object to sub class reference variable then you will get class cast Exception
- In order to do that we need to convert super class Object to sub class Object explicitly

- jvm will not do downcasting developer has do that

Demo :

```
class A{
int i=5;
}
class B{
int f=45;
}
class Demo{
public static void main(String[] args){
A a=new B();   //down casting
}
```

InstanceOf Operator

- Instance operator hold the address of the current Object

- If we want to check any Object belong to the current class or not then you can use instanceOf operator

```
Demo ;
class str{
public static void main(String[] args){
str s=new str();
if(s instanceOf str){
System.out.println(s);
}
else{
System.out.println("no");
}
}
}
```

Packages in Java

Definition:
A package is a collection of classes, interfaces, sub packages and annotations.

Uses:

- Code reusability.
- To avoid naming collision i.e. to maintain more than one class with same but different functions.

Root package:

J2SE	java
J2EE	javax

Packages in Java (contd.)

create userdefined package:

1. To create a userdefined package we must write “package package_name” in our code.
2. Then during compilation we must use “-d . “.

Syntax:

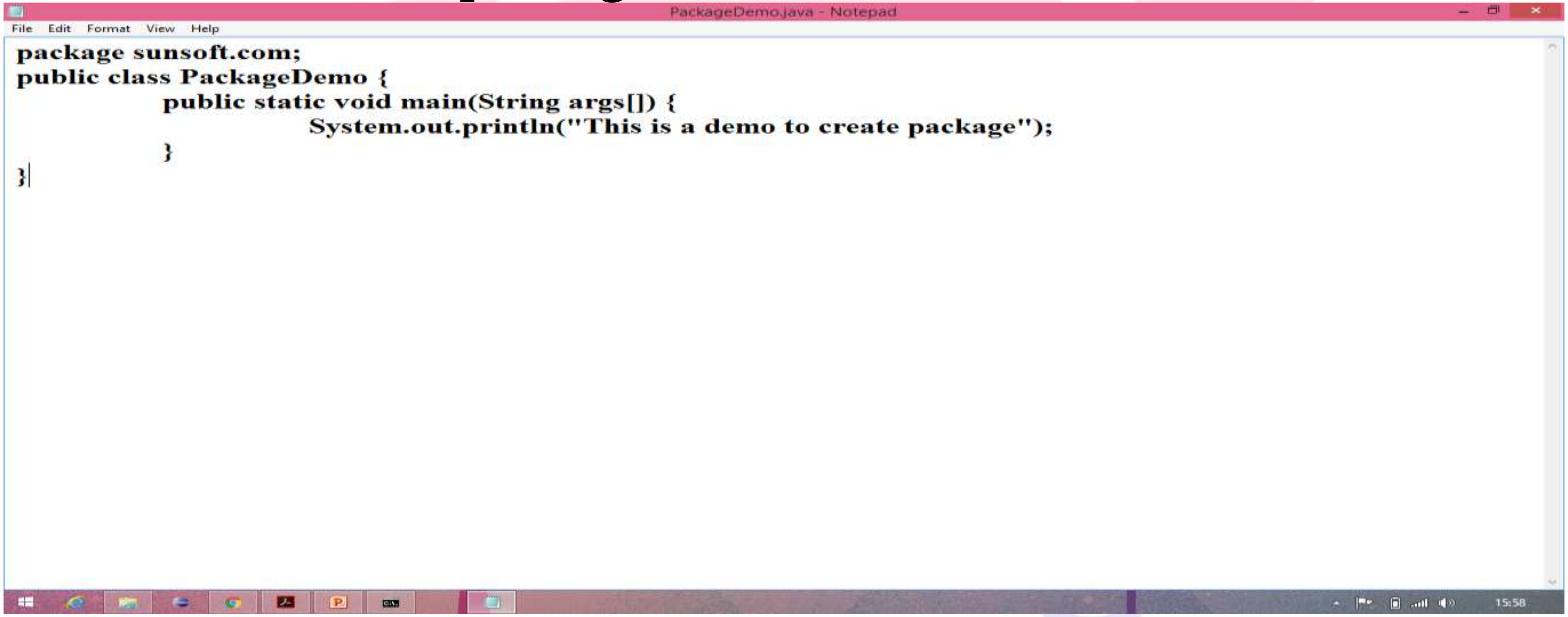
```
javac –d . FileName
```

Note: While creating a package we must follow the order given below.

$$\begin{tikzcd}\textbf{package} \arrow[r] & \textbf{import} \arrow[r] & \textbf{class}\end{tikzcd}$$

Packages in Java (contd.)

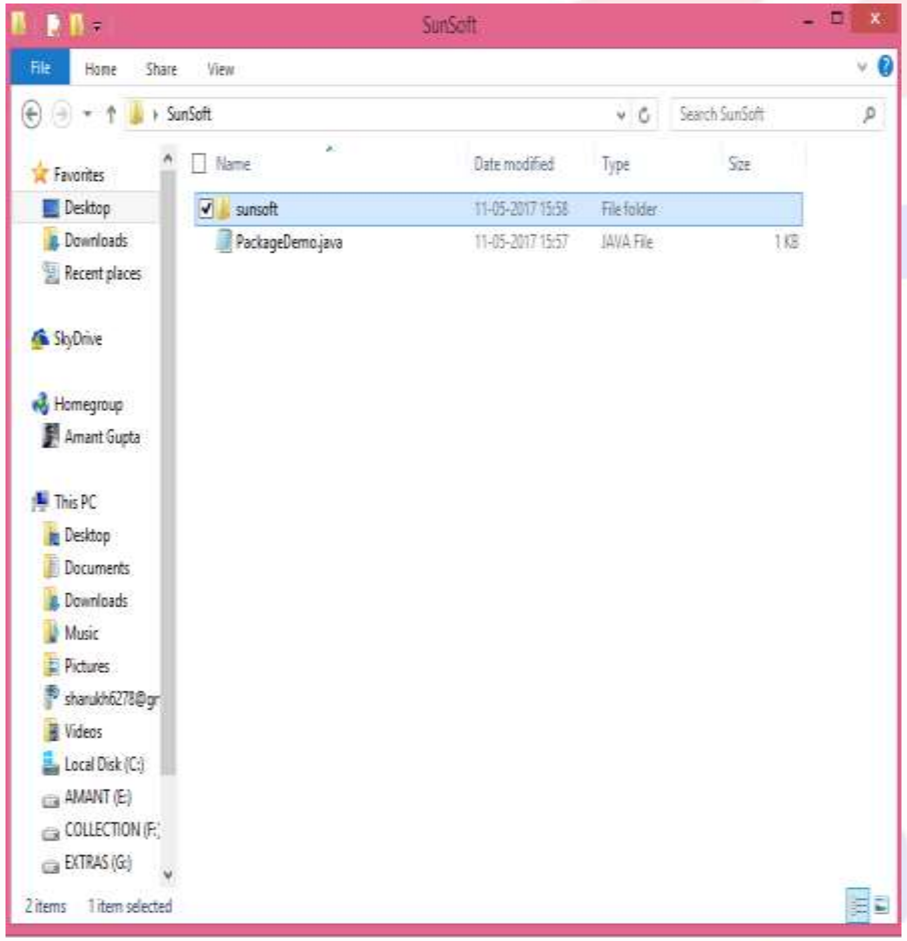
create userdefined package demo:



```
package sunsoft.com;
public class PackageDemo {
		public static void main(String args[]) {
				System.out.println("This is a demo to create package");
		}
}
```

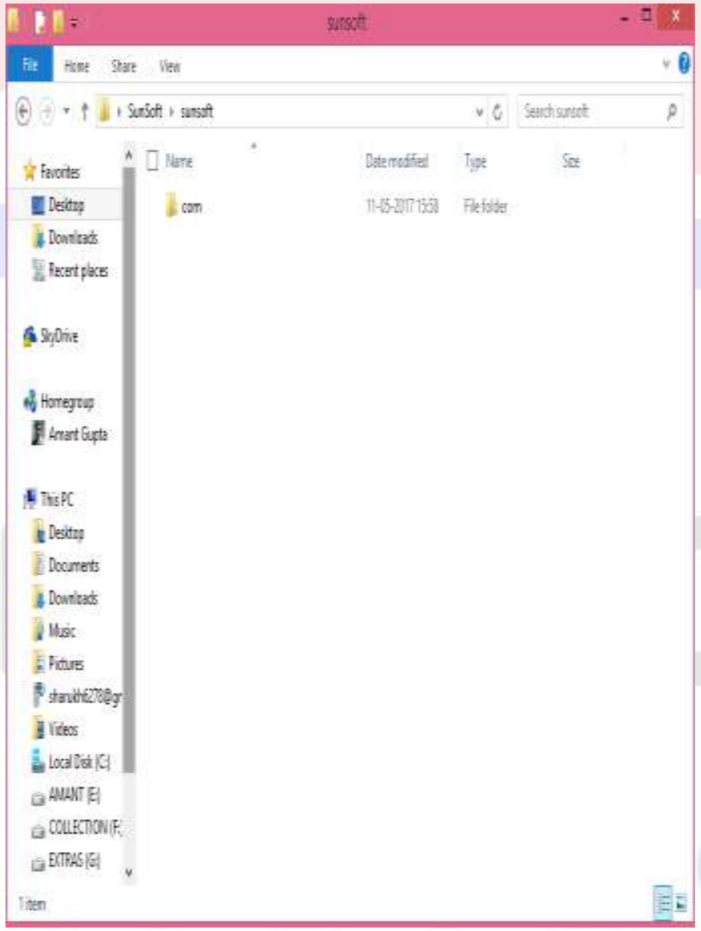
Packages in Java (contd.)

create userdefined package demo:



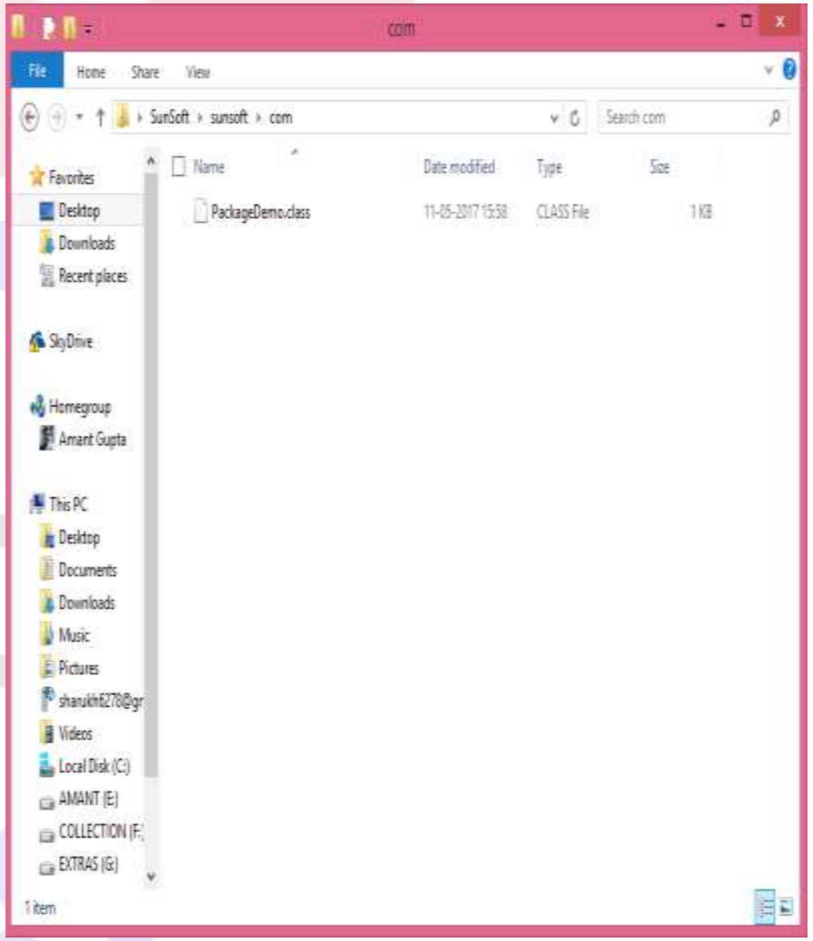
Name	Date modified	Type	Size
sunsoft	11-05-2017 15:58	File folder	
PackageDemo.java	11-05-2017 15:57	JAVA File	1 KB

2 items 1 item selected



Name	Date modified	Type	Size
com	11-05-2017 15:58	File folder	

1 item

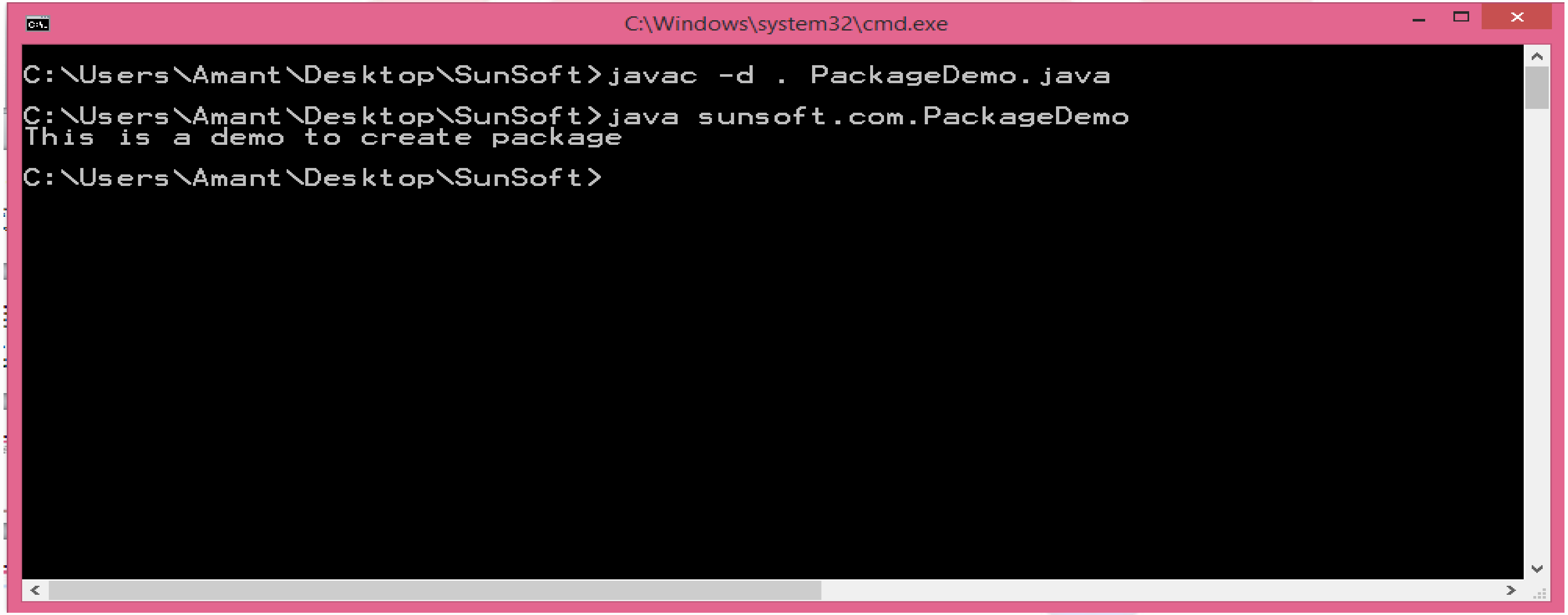


Name	Date modified	Type	Size
PackageDemo.class	11-05-2017 15:58	CLASS File	1 KB

1 item

Packages in Java (contd.)

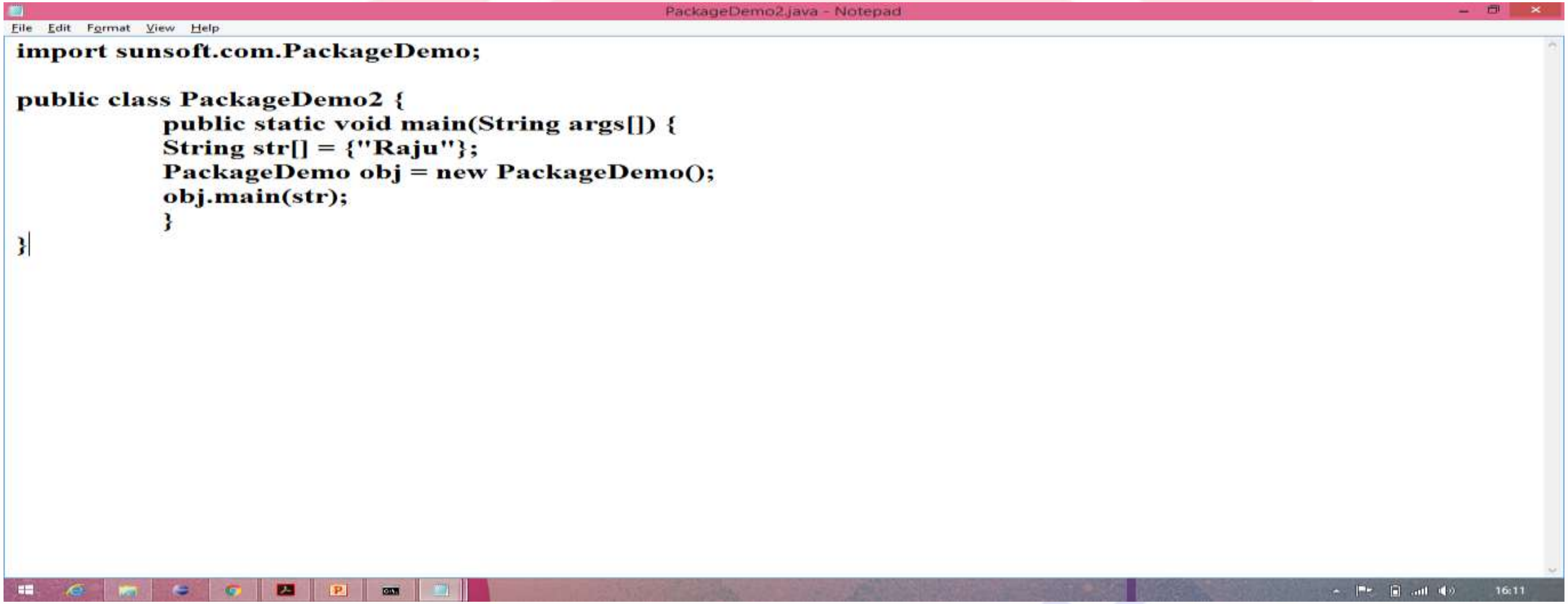
create userdefined package demo:



```
C:\Users\Amant\Desktop\SunSoft>javac -d . PackageDemo.java

C:\Users\Amant\Desktop\SunSoft>java sunsoft.com.PackageDemo
This is a demo to create package

C:\Users\Amant\Desktop\SunSoft>
```

Packages in Java (contd.)

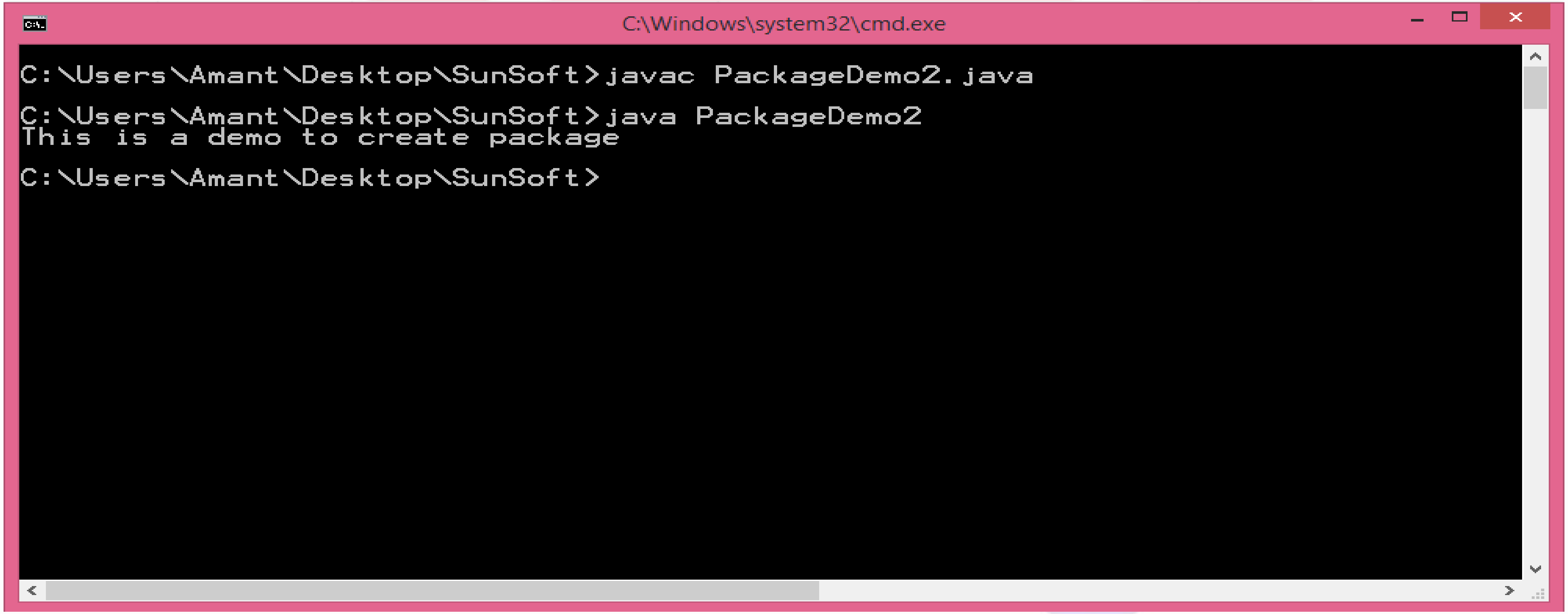
Importing the userdefined packages:



```
import sunsoft.com.PackageDemo;

public class PackageDemo2 {
                public static void main(String args[]) {
                String str[] = {"Raju"};
                PackageDemo obj = new PackageDemo();
                obj.main(str);
                }
}
```

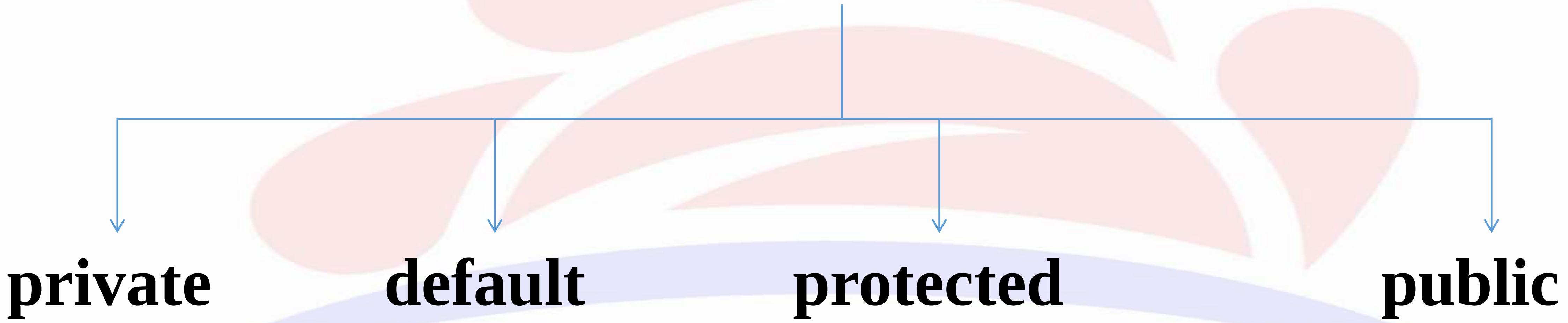
Packages in Java (contd.)

Importing the userdefined packages:


C:\Windows\system32\cmd.exe

```
C:\Users\Amant\Desktop\SunSoft>javac PackageDemo2.java

C:\Users\Amant\Desktop\SunSoft>java PackageDemo2
This is a demo to create package

C:\Users\Amant\Desktop\SunSoft>
```

Access specifier



Access specifier (contd.)

1. **private scope:** the scope of private access specifier is throughout the class.
2. **default scope:** the scope of default access specifier is throughout the package.
3. **protected scope:** the scope of protected access specifier is throughout the class as well as the sub classes of that class.
4. **public scope:** the scope of public access specifier is global i.e. it can be globally accessed.

Note:

- ‘private’ and ‘protected’ access specifier cannot be used by the outer class. They can only be used by the members of the class.
- there is no keyword used for default access modifier.

Scanner class

- Scanner class is used to take input from keyboard.
- Scanner class breaks its input into tokens using a delimiter pattern, which by default whitespace.
- Java Scanner class extends Object class and implements java.util.Iterator and java.io.Closeable interfaces.

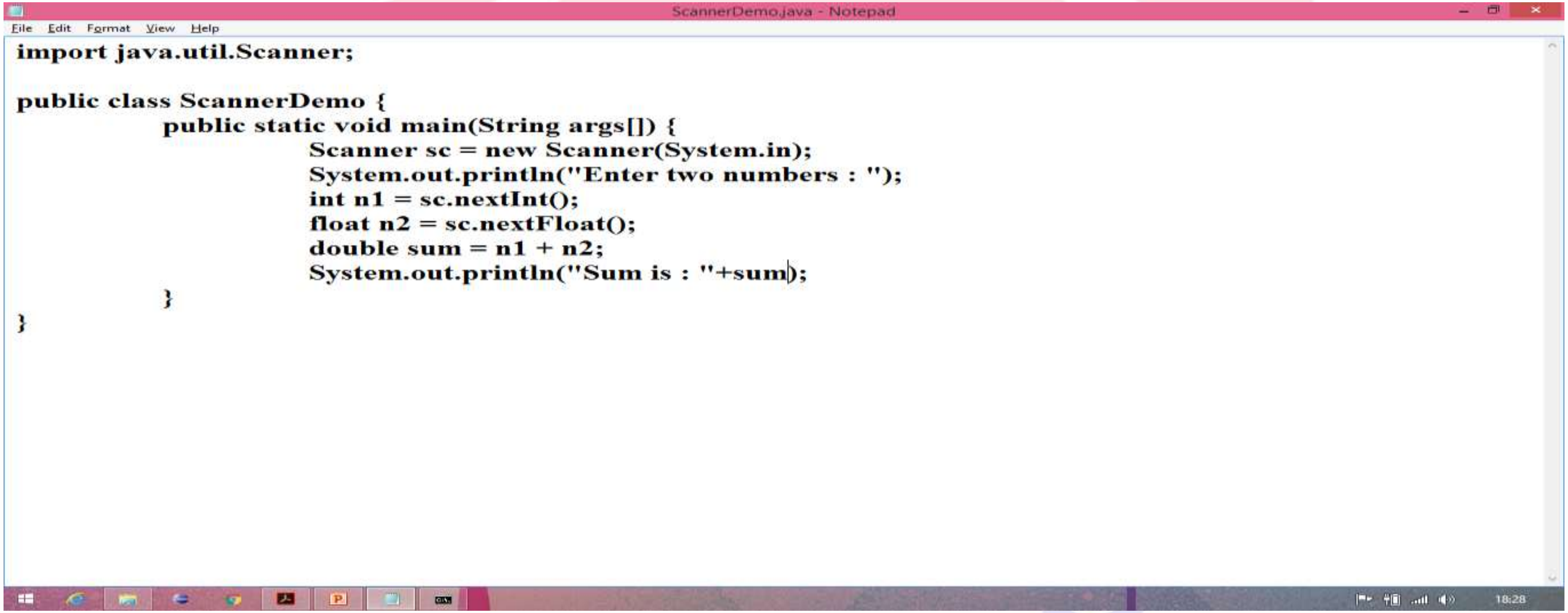
Syntax to import Scanner class:

```
import java.util.*;
import java.util.Scanner;
```

Note: it is not advised to use **import java.util.*;** as it imports all the classes present in scanner class.

Scanner class (contd.)

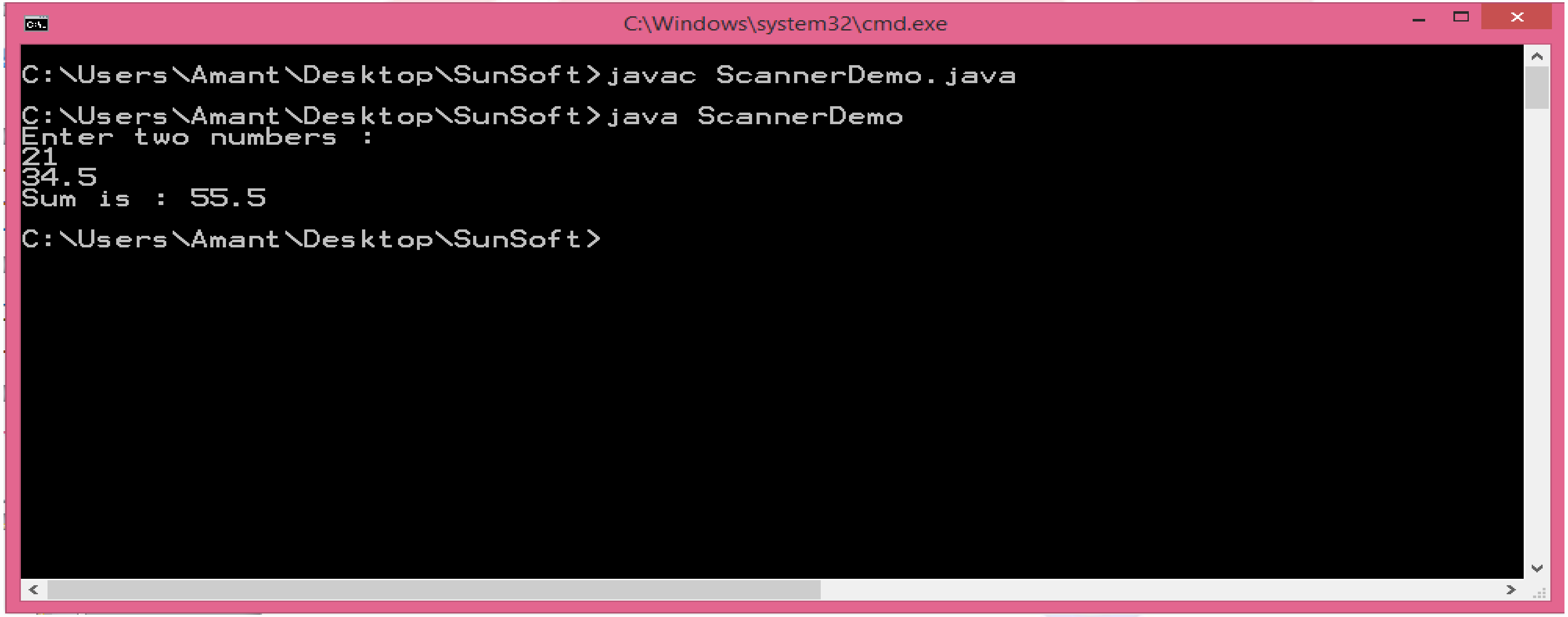
Demo:



```
import java.util.Scanner;

public class ScannerDemo {
		public static void main(String args[]) {
				Scanner sc = new Scanner(System.in);
				System.out.println("Enter two numbers : ");
				int n1 = sc.nextInt();
				float n2 = sc.nextFloat();
				double sum = n1 + n2;
				System.out.println("Sum is : "+sum);
		}
}
```

Scanner class (contd.)

Demo:



```
C:\Users\Amant\Desktop\SunSoft>javac ScannerDemo.java

C:\Users\Amant\Desktop\SunSoft>java ScannerDemo
Enter two numbers :
21
34.5
Sum is : 55.5

C:\Users\Amant\Desktop\SunSoft>
```

Exception

- Unwanted and unexpected event occurs that disturb the normal flow of execution is called exception

ex: arithmetic Exception, null pointer Exception, array indexOutOfBound Exception

- Main Objective of Exception Handling is to graceful termination of the program

- Handling Exception does not mean repairing Exception it is to provide alternative to continue rest of the program normally.

ex : if you trying to access online file in case it is not available then file not found Exception will throw by jvm and terminate the program, to avoid abnormal termination of the program we need to provide local file to continue rest of the execution of the program normal.

Types of exception

1)Checked Exception : the Exception checked by the Compiler

- Checked exception should be handle before compilation, with the help of try or throws key word otherwise it will raise Exception on compilation time

2)Unchecked Exception : run time Exception which will not be checked by the compiler

try Block

- Risky code means the code which raise exception should be write in try block
- try block is used handle both check and unchecked Exception
- If exception raises in try block then control will transfer to catch block where Exception handling code will written
- And control will not come back it will continue from catch block itself

catch Block

- In this block the Exception handling code will written
- If we did not handle the Exception then default Exception handler will handle the Exception and terminate the execution
- Catch Block always followed by try

finally Block

- Finally Block will always execute regarding of raising Exception
- Finally always with be try or try catch block
- Finally block mostly used to close the connection witch are open

Differences between final, finally, finalize()

final

- final is a key word applicable to class variable , method
- When you declare variable as final then its value cant be change
- When you declare method as final then you can not override that method
- When you declare class as final then you can not inherit that class

Differences between final, finally, finalize()

finally

- ➢Finally is a block always associated with try catch block
- ➢This block is used to write clean up code for open connection

finalize()

- ➢Is a method of Object whose work is to close the connection.
- ➢finalize() is called after the Object is destroyed.

Throw key word

- Throw key word is used to throw Exception Explicitly
- It is used in custom Exception to throw exception

  ex : throw io Exception
- We can throw checked and unchecked both Exception with throw key word

  ex : arithmetic Exception

Throws key word

- Throws is a key word used to handle checked exception

  ex: io Exception
- Throws key word does not handle unchecked exception for that we need to use try catch block to handle run time Exception
- Using throws key word when run time Exception is raising is useless

Exception hierarchy

- Throwable class act as root class for all java Exception
- Throwable class define two more child classes

    1.Exception 2.Error

- Exception are mostly caused by our program and these can be recoverable
- Error are mostly caused by lack of system resources and cannot be recoverable and all Error and its subclass are unchecked

Differences between checked and unchecked Exception

checked Exception

- This exception checked by compiler
- Throws key word and try catch block is used to handle this Exception
- Ex :

  io exception

unchecked Exception

- Compiler will not check this Exception
- Try catch block are used to handle this Exception
- Ex
- Arithmetic Exception

Handling all type of Exception

- We can handle all type of Exception in catch block
- Write number of catch block as number of Exception you want to handle
- Order is important for that you can give catch(Exception e) in last
- You can write only one catch block to handle many Exception by using || when handling code is similar for all Exception

Custom Exception

- User defined Exception is called as custom Exception

  Ex : invalid password or user name or insufficient fund etc.
- We can create both checked and uncheckedException
- For that if you want create checked Exception then your class must Extends Exception class

- If you want create run time Exception then you need to extends your class with RuntimeException

  syntax

  class ClassName extends RuntimeException

1.Can we write try without catch

No try always comes with catch block

2.Can we write catch without try

No catch always followed by try block

3.Can we write first catch and then try

No catch always followed by try block

4.Can we write try then finally

Yes try should either with catch or finally

5.Can we write Finally then try

No finally should be below try or catch block

6.Can we write

Try with multiple catch

yes

7.Can we write

Try catch inside try block

yes

Predefined packages

- Predefined packages are inbuilt package which contain useful classes interface and method

  a. java.lang
  b. java.io
  c. java.util

lang package

- This is predefined package that is java.lang
- By default lang package in imported in java
- lang package contain some useful classes and interface that are required to develop the simple java program
- Object , class String , StringBuffer , StringBuilder , Math…..etc. class are belong to the java.lang package

io package

- This is predefined package that is java.io
- Manually java.io package need to be imported
- io package contain some useful class and interface that are required to handle files
- File , serialization, Externalization , …..etc. class and interface are belong to the java.io package

util package

- This is predefined package that is java.util
- Manually java.util package need to be imported
- util package contain some useful class and interface that are required for utility purpose
- Arrays ,Collection, Collections, Date, …..etc. class and interface are belong to the java.io package

Multi-threading

java.lang.Thread

- **Program :**

  Set of Instruction to Perform a specific task
- **Process :**

  Time taken to Execute the Code
- **Processor :**

  it executes the Code
- **Thread :**

  Small execution of Code within process
  - Thread is light weight Process
    - Java is Thread based language
  - C and C++ are process based language

- **Multi-tasking :**

  executing several task simultaneously

- **Multi-processing :**

  executing several task simultaneously where each task is independent of each other

  ex: typing program listening music downloading file simultaneously

➢Multi-threading :

executing several task simultaneously where each task is independent of each but they are part of the same program

Ex: executing 2 separate block of the same program simultaneously

➢Thread scheduler :

is component of jvm whose work is to schedule the thread execution

Some common method of Thread class

currentThread() :

gives current thread details

sleep() :

used to sleep the thread

suspend() :

used to suspend the thread

resume() :

used to resume the thread

wait():

used to wait the thread

notify() :

notify the thread which is waiting

notifyAll() :

notify all thread which are waiting

start() :

used to start new thread

stop() :

stop the thread

join() :

thread waits until another thread done its execution

setName() **:**

used to set the name to thread

getName() **:**

used to get the name of thread

setPriority() **:**

used to set the Priority to thread

getPriority() **:**

used to get the priority

activeCount() **:**

give the number of active thread

isDaemon() **:**

returns boolean value

setDaemon() **:**

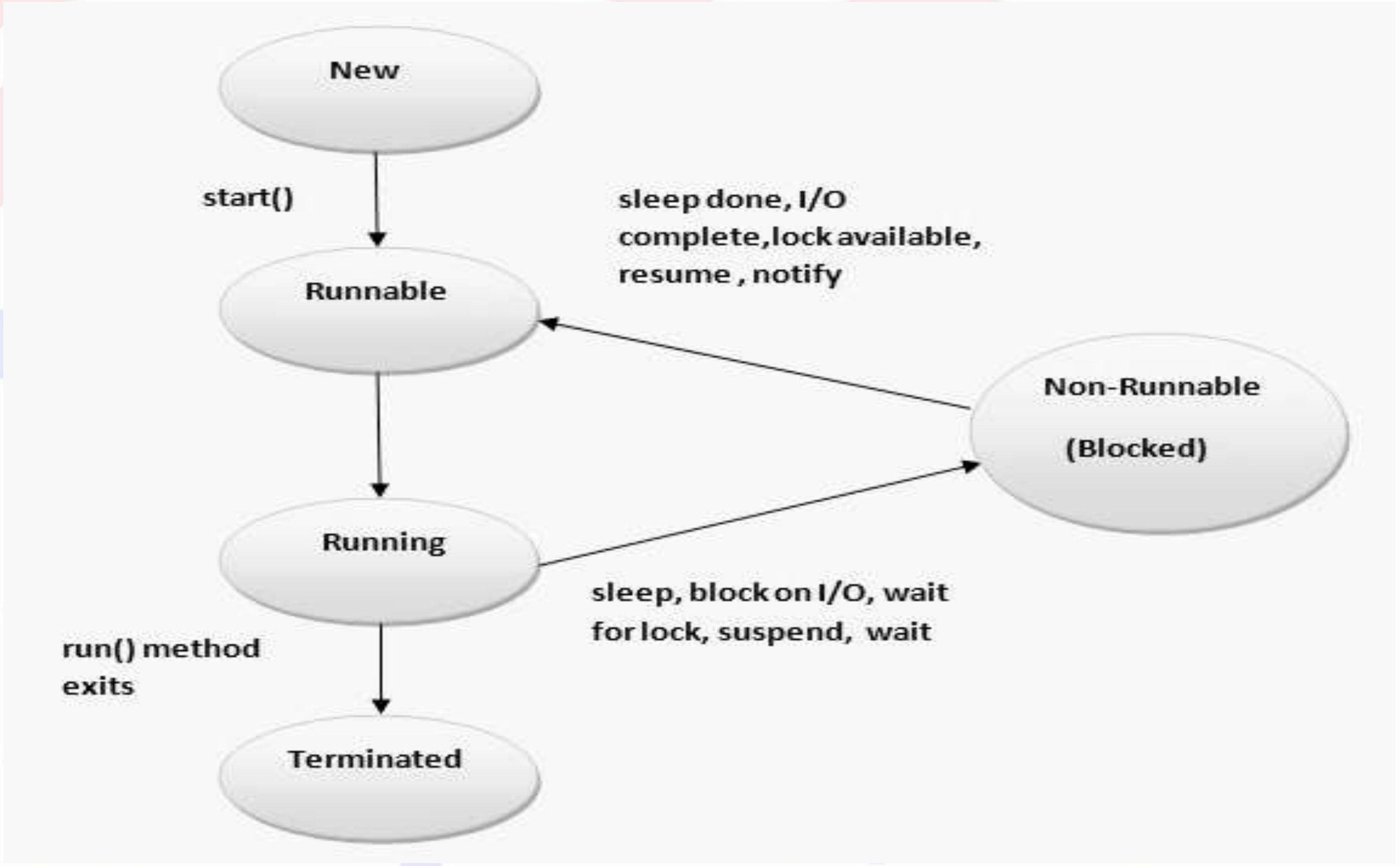
used to set thread Daemon

yield() **:**

checks any thread waiting which are having priority value greater then this thread if available it will give chance to execute the thread and goes into waiting state

Thread life cycle

1.New
2.Runnable
3.Running
4.Non-Runnable (Blocked)
5.Terminated(dead)


New
start()
Runnable
sleep done, I/O complete,lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

1. **New :**

    new thread is created before calling start ()

2. **Runnable :**

    thread is runnable state after calling start()(means its ready to run)

3. **Running :**

    Thread is in running state if thread scheduler allocate the thread to the processor

4. **Non Runnable(Blocked) :**

    alive thread but not in running state

5. **Terminated(Dead) :**

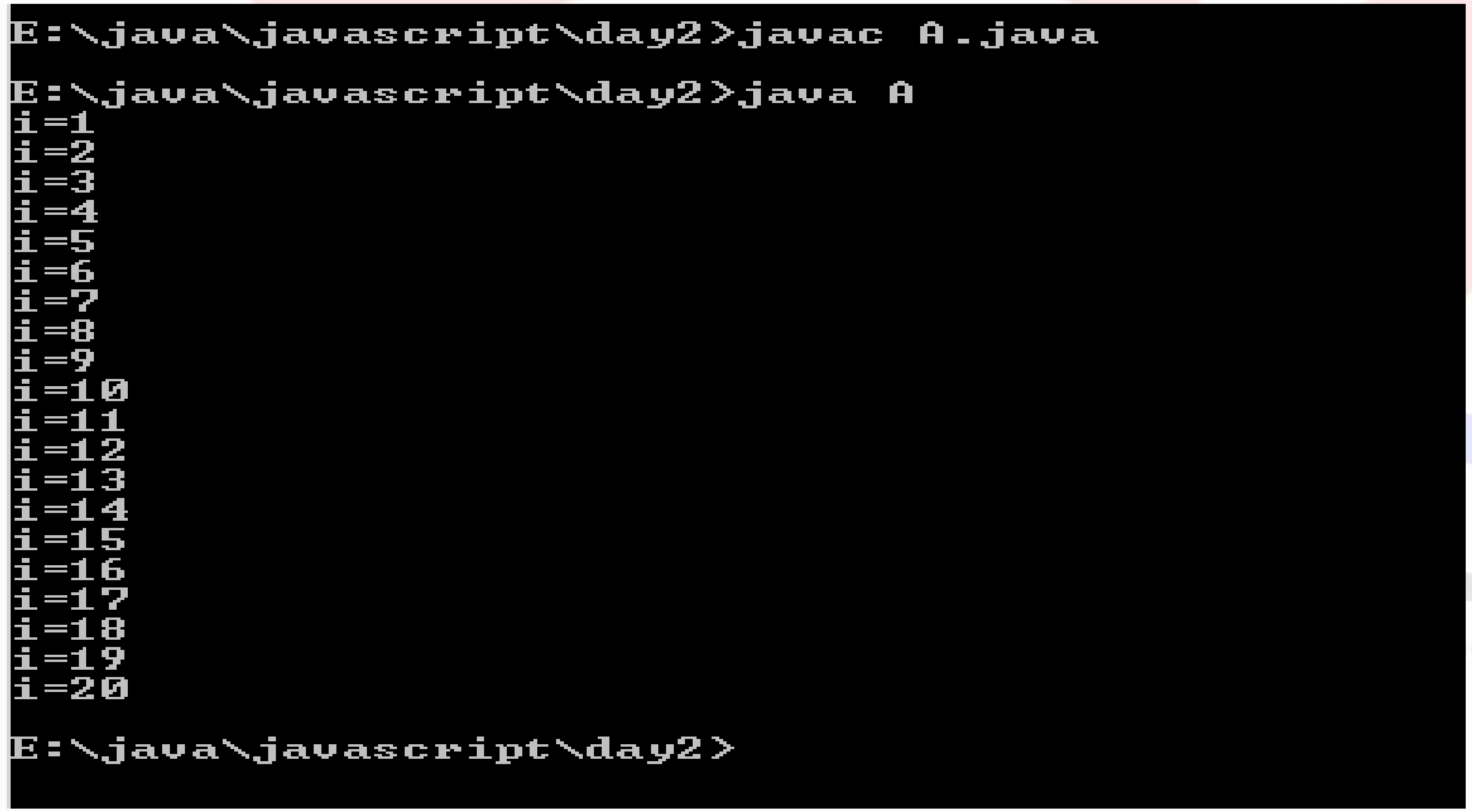
    thread is in dead state(Terminated)

- Thread is pre-defined class belong to the package java.lang.Thread

Demo :normal program

```
            class A
{
            public static void main(String[] args)
            {
                        for(int i=1;i<=20;i++)
                        {
                        System.out.println("I ="+i);
                        }
            }
}
```

Output : 1st and 2nd run



```
E:\java\javascript\day2>javac A.java

E:\java\javascript\day2>java A
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10
i=11
i=12
i=13
i=14
i=15
i=16
i=17
i=18
i=19
i=20

E:\java\javascript\day2>
```

- **In 2 ways we can achieve multi-threading**

1.extending Thread class

2.implementing runnable interface

1.extending Thread class

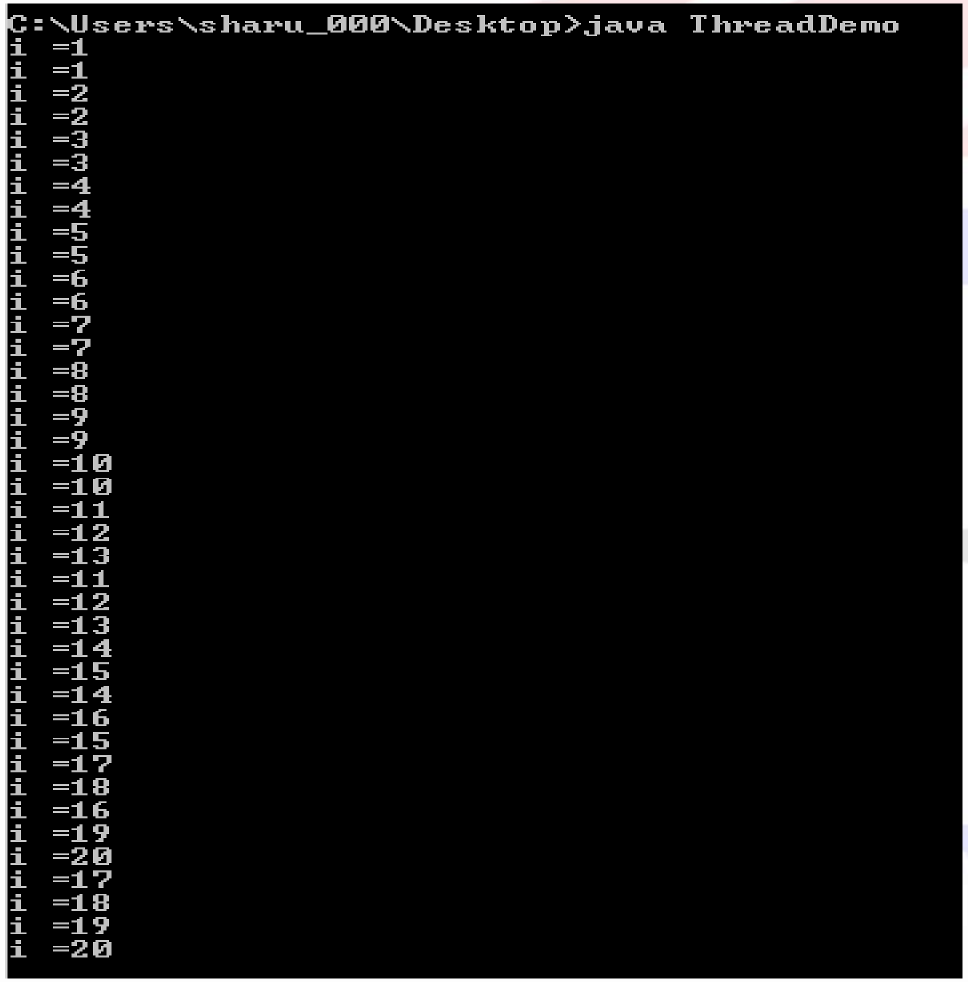
- we need to extends Thread class and override run()
- Create Object of our class
- call start() on object of our class
- Calling start() will create new thread and automatically invoke run()

Demo :

```
class  A extends Thread
{
            public void run()
            {
                        For(int  i=1;i<=20;i++)
                        {
                                    System.out.println("i ="+i)         ;
                        }
            }
}
class ThreadDemo
{
            Public static void main(String[] args)
            {
                        A  a1=new A();
                        A a2=new  A();
                        a1.start();
                        a2.start();
            }
}
```

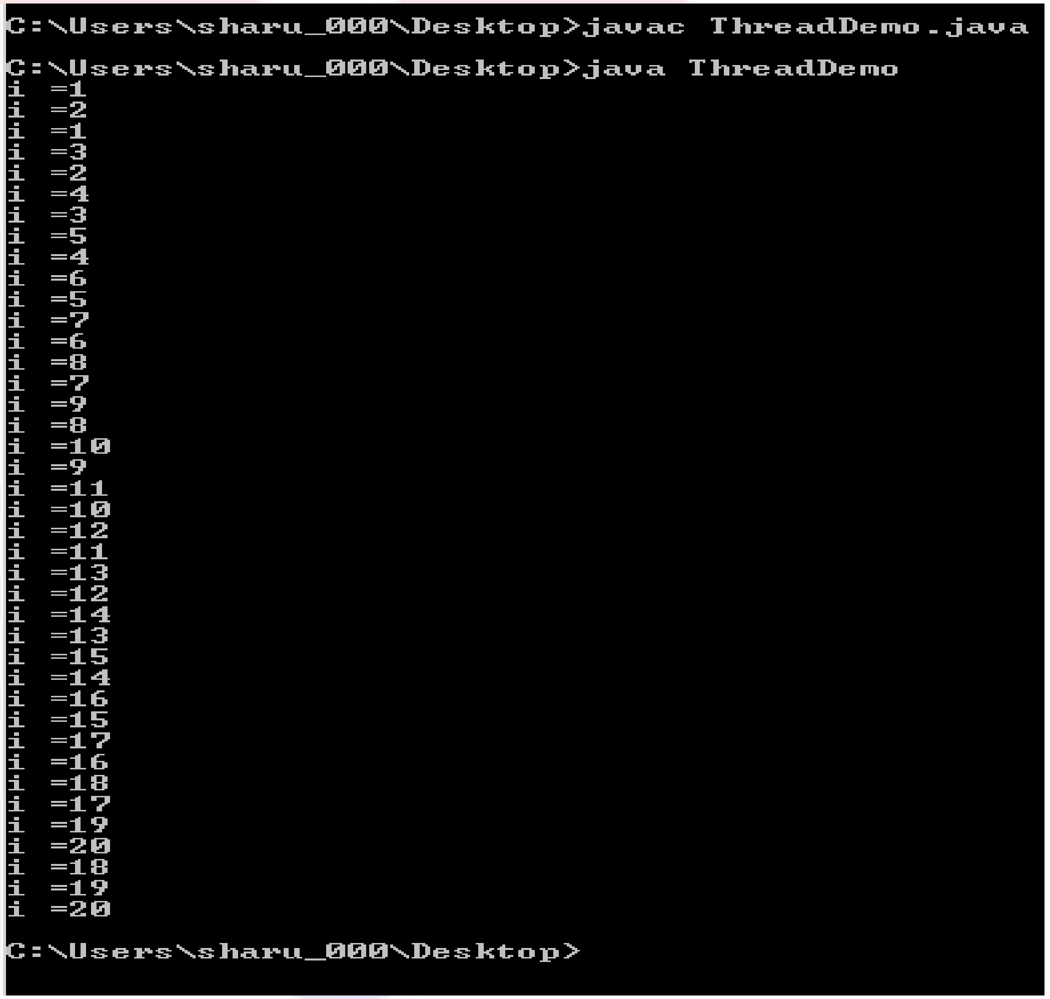
Output :

1st run



```
C:\Users\sharu_000\Desktop>java ThreadDemo
i =1
i =1
i =2
i =2
i =3
i =3
i =4
i =4
i =5
i =5
i =6
i =6
i =7
i =7
i =8
i =8
i =9
i =9
i =10
i =10
i =11
i =12
i =13
i =11
i =12
i =13
i =14
i =15
i =14
i =16
i =15
i =17
i =18
i =16
i =19
i =20
i =17
i =18
i =19
i =20
```

2nd run



```
C:\Users\sharu_000\Desktop>javac ThreadDemo.java

C:\Users\sharu_000\Desktop>java ThreadDemo
i =1
i =2
i =1
i =3
i =2
i =4
i =3
i =5
i =4
i =6
i =5
i =7
i =6
i =8
i =7
i =9
i =8
i =10
i =9
i =11
i =10
i =12
i =11
i =13
i =12
i =14
i =13
i =15
i =14
i =16
i =15
i =17
i =16
i =18
i =17
i =19
i =20
i =18
i =19
i =20

C:\Users\sharu_000\Desktop>
```

II. Implementing runnable interface

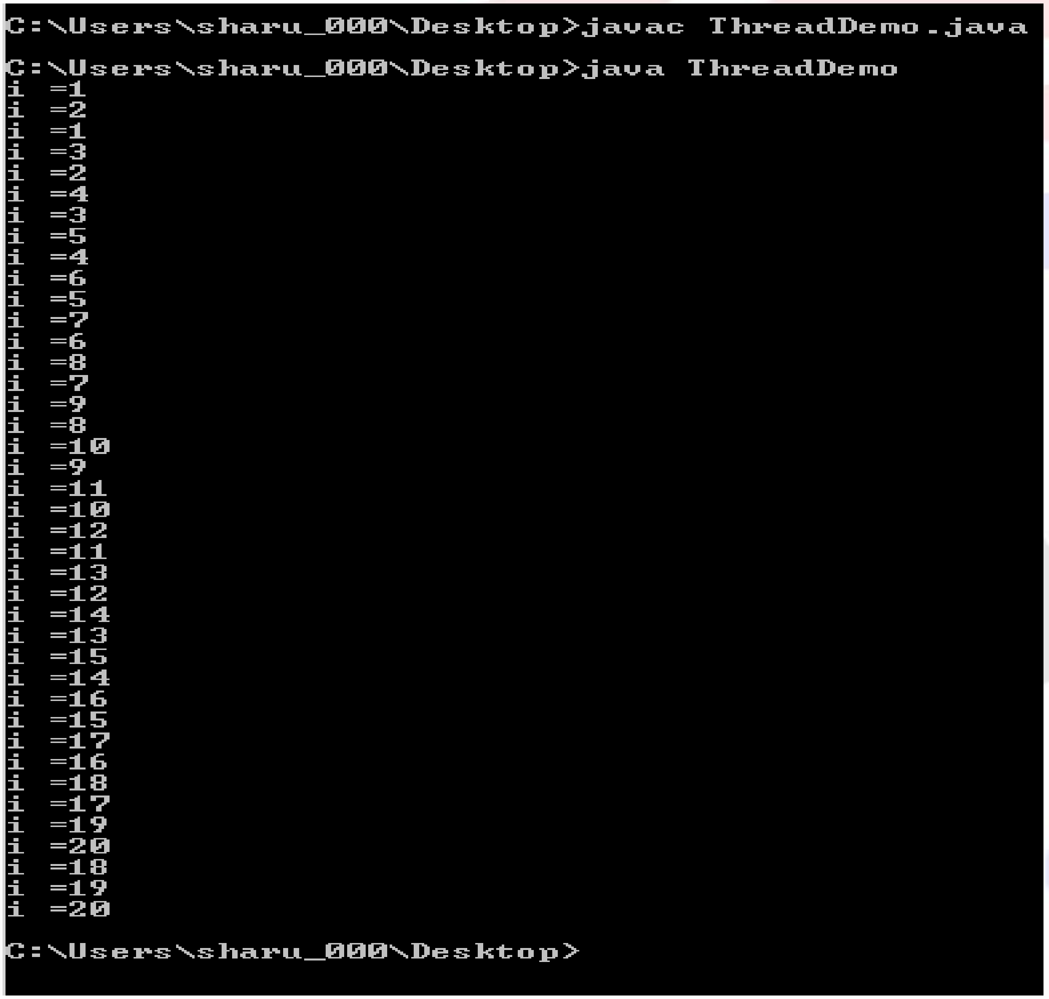
- ➢Runnable is an interface Available in java.lang package
- ➢To create thread we need to implements **runnable interface** to our class and override **run() method**
- ➢create Object of the class ,then create Thread Object by passing your class Object as parameter and call start method on Thread Object reference

Demo :

```
class A implements runnable
{
        public void run()
        {
                for(int i=1;i<=20;i++)
                {
                        System.out.println(Thread.currentThread.currentThread.getName()+i);
                }
        }
}
class ThreadDemo
{
        public static void main(String[] args)
        {
                A a1=new A();
                Thread d1=new Thread(a1);
                Thread d2=new Thread(a1);
                d1.start();
                d2.start();

        }
}
```

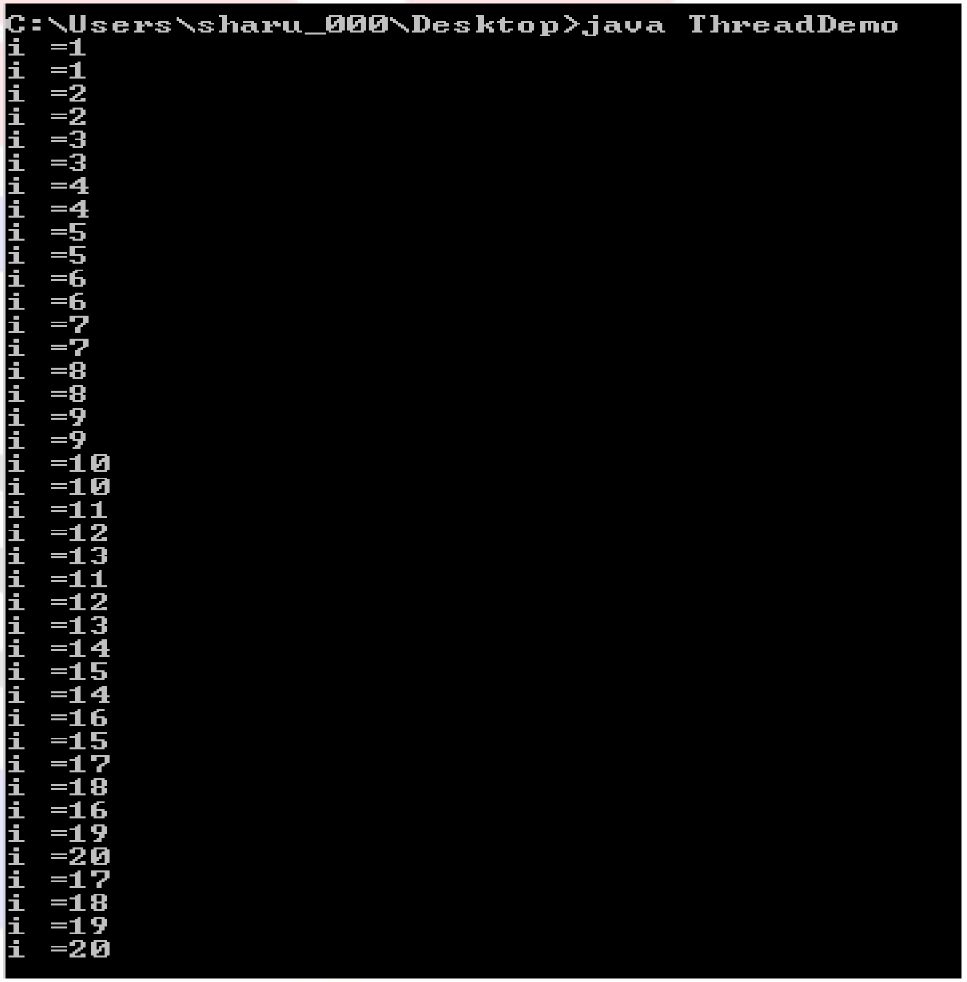
Output :

1st run



```
C:\Users\sharu_000\Desktop>javac ThreadDemo.java

C:\Users\sharu_000\Desktop>java ThreadDemo
i =1
i =2
i =1
i =3
i =2
i =4
i =3
i =5
i =4
i =6
i =5
i =7
i =6
i =8
i =7
i =9
i =8
i =10
i =9
i =11
i =10
i =12
i =11
i =13
i =12
i =14
i =13
i =15
i =14
i =16
i =15
i =17
i =16
i =18
i =17
i =19
i =20
i =18
i =19
i =20

C:\Users\sharu_000\Desktop>
```

2nd run



```
C:\Users\sharu_000\Desktop>java ThreadDemo
i =1
i =1
i =2
i =2
i =3
i =3
i =4
i =4
i =5
i =5
i =6
i =6
i =7
i =7
i =8
i =8
i =9
i =9
i =10
i =10
i =11
i =12
i =13
i =11
i =12
i =13
i =14
i =15
i =14
i =16
i =15
i =17
i =18
i =16
i =19
i =20
i =17
i =18
i =19
i =20
```

Difference between start() and run()

- Calling start() will create new Thread ,calling run() will not create any thread it will execute as normal method
- You can’t call start method twice on same thread it will throw Exception
- run() will automatically invoke when we call start()

Thread Priorities

➢Based on thread priority processor will execute the thread

➢Java threads priority is range between 1 to 10

1. public static int MIN_PRIORITY=1

2.public static int NORM_PRIORITY=5

3.public static int MAX_PRIORITY=10

➢The thread which is having MAX_PRIORITY will execute first

➢The threads which is having MIN_PRIORITY will execute last

Thread Priorities

- ➢Default priority is 5 for main thread
- ➢Priority will be inherited from parent thread to child thread means if you not given priority to your nondaemon thread it will inherit the priority from main thread
- ➢setPriority() is used to set the priority for thread
- ➢getPriority() will return priority of thread that is int value

Daemon Thread

- ➢Daemon threads are those thread which provides support for other thread or non-daemon Threads
- ➢Daemon threads will run in back ground

Ex : Garbage collector, main Thread

- ➢Daemon Thread priority will less then non-daemon thread but based on situation jvm will increase their

  thread priority

- if their is no thread then jvm will terminate the daemon thread
- Methods for Daemon threads

  1. public void setDaemon(boolean status)

     used to set thread as Daemon Thread

  2. public boolean isDaemon()

     return boolean value checks the thread is Daemon or not
- To make thread as daemon it should set Daemon before it starting otherwise illegalthreadstateException will throw

Thread Pool

- thread pool is group of thread or worker threads which are waiting for job
- Advantage of thread pool is we can reuse the thread as many time hence performance will be increases
- In thread pool a group of fixed number of threads create and a thread from thread pool will be pulled and assign the job
- After completion of the job the thread will go back into the thread pool

- thread pools are used in jsp and servlet where container creates a thread pool to process the request
- The following code will create thread pool of 10 threads

```
ExecutorService executor = Executors.newFixedThreadPool(10)
```

Thread group

- Java provides a convenient way to group multiple threads in a single object
- Thread group can be used to suspend resume interrupt group of thread by single method
- Thread group implemented by *java.lang.ThreadGroup*

Constructors of ThreadGroup class

1. ThreadGroup(String name)

    create thread group of given name

2. ThreadGroup(ThreadGroup parent, String name)

    create thread group of given parent group and name

Methods in ThreadGroup

1. int activeCount()

    returns no. of threads running in current group

2. int activeGroupCount()

returns a no. of active group in this thread group

3. void destroy()

Destroy the thread group and all its subgroup

3. String getName()

returns the name of the group

4. ThreadGroup getParent()

returns the parent of this group

5. void interrupt()

interrupt all the thread of the thread group

Synchronization

- Synchronization is key words applicable on methods and blocks
- Synchronization will allow to execute only one thread at time To prevent thread interference
- Synchronization also prevent consistency problem

ex :banking , seat booking

1.Can we start a thread twice ?

No (java.lang.IllegalThreadStateException)

2.Can we overload the run() ?

Ans: : yes, but start method always look for default run() if not available then no output)

3.Can we override start() ?

Ans :yes, but it will invoked as normal method

4.What will happen if we call run() directly ?

Ans : it will execute as normal method

Object class

- Object is a predefined class in java.lang package.
- Object class is the root class of all the classes in java.lang package.
- Object is the default super class of all the classes in java.lang package.
- When we define any class without extending any other class then Object class will be direct super class otherwise Object class will be indirect super class.
- The members available in this class can be referred with any type of reference like class, interface, array etc.
- Object class reference variable can hold any type of object.

Object class

Some useful methods:

- ➢ public final native Class getClass()
- ➢ public native int hashCode()
- ➢ public String toString()
- ➢ public boolean equals(Object)
- ➢ protected native Object clone()
- ➢ protected void finalize()
- ➢ public final native void notify()
- ➢ public final native void notifyAll()
- ➢ public final void wait(

String, StringBuffer, StringBuilder

String:

- ➢ String class is present in java.lang package.
- ➢ String objects are immutable.
- ➢ When you try to modify the contents of object then new object is created.
- ➢ String class is final i.e. it cannot be extended.
- ➢ String class implements the following interfaces:
  - java.io.Serializable
  - java.lang.Comparable
  - java.lang.CharSequence
- ➢ String object can be created in two ways:
  - Without using new keyword.
  - By using new keyword.

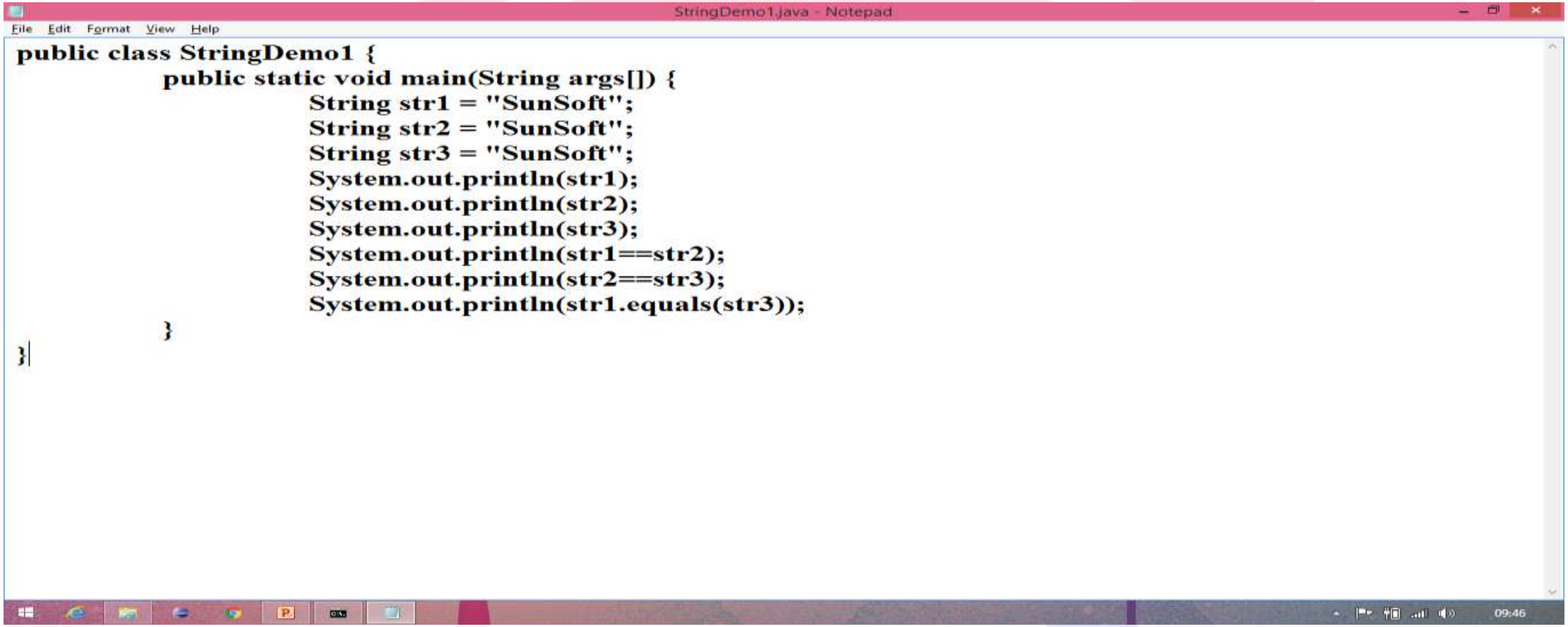
String, StringBuffer, StringBuilder (contd.)

Creating String object without new keyword:

- JVM allocates the memory for the String reference variable.
- JVM verifies the String literal in the String Constant Pool.
- If String literal is not available in the String Constant Pool then it creates new String object inside the String Constant pool and the newly created String object address will be assigned to String reference variable.
- If String literal is available then existing String object address will be assigned to String reference variable.

String, StringBuffer, StringBuilder (contd.)

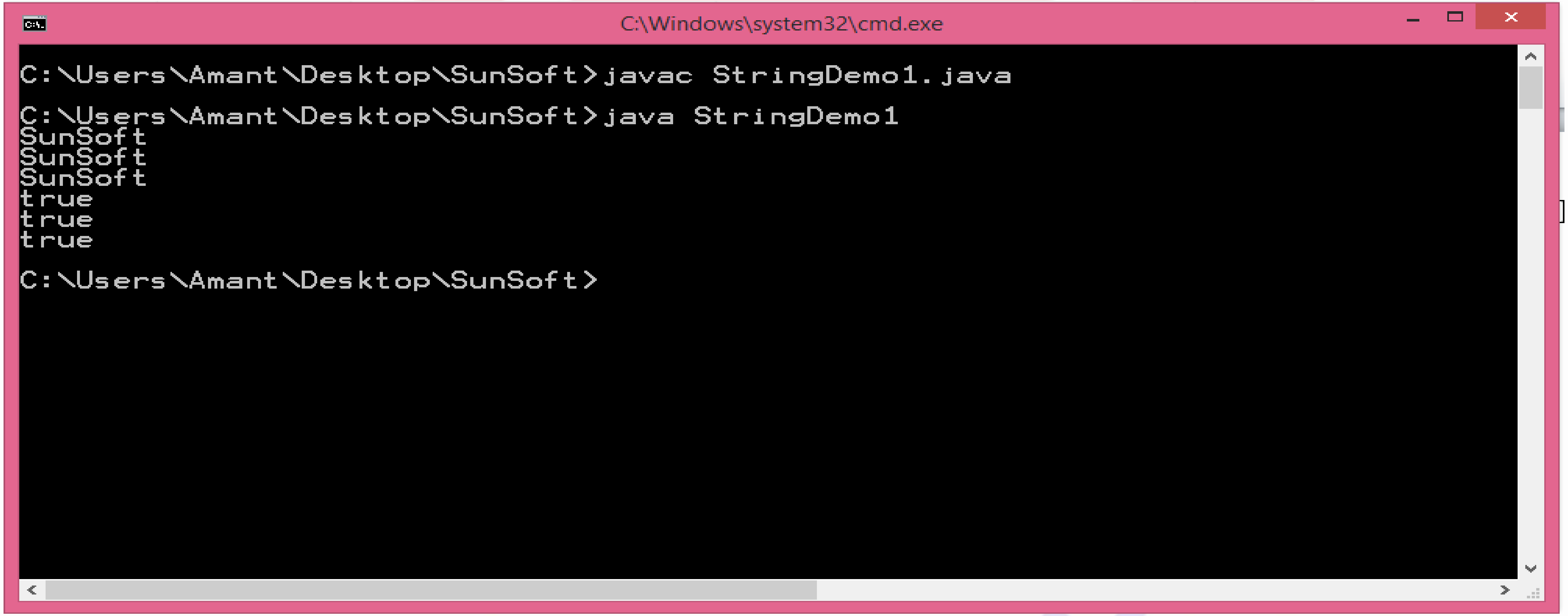
Creating String object without new keyword Demo:



```
public class StringDemo1 {
            public static void main(String args[]) {
                        String str1 = "SunSoft";
                        String str2 = "SunSoft";
                        String str3 = "SunSoft";
                        System.out.println(str1);
                        System.out.println(str2);
                        System.out.println(str3);
                        System.out.println(str1==str2);
                        System.out.println(str2==str3);
                        System.out.println(str1.equals(str3));
            }
}
```

String, StringBuffer, StringBuilder (contd.)

Creating String object without new keyword Demo:



```
C:\Users\Amant\Desktop\SunSoft>javac StringDemo1.java

C:\Users\Amant\Desktop\SunSoft>java StringDemo1
SunSoft
SunSoft
SunSoft
true
true
true

C:\Users\Amant\Desktop\SunSoft>
```

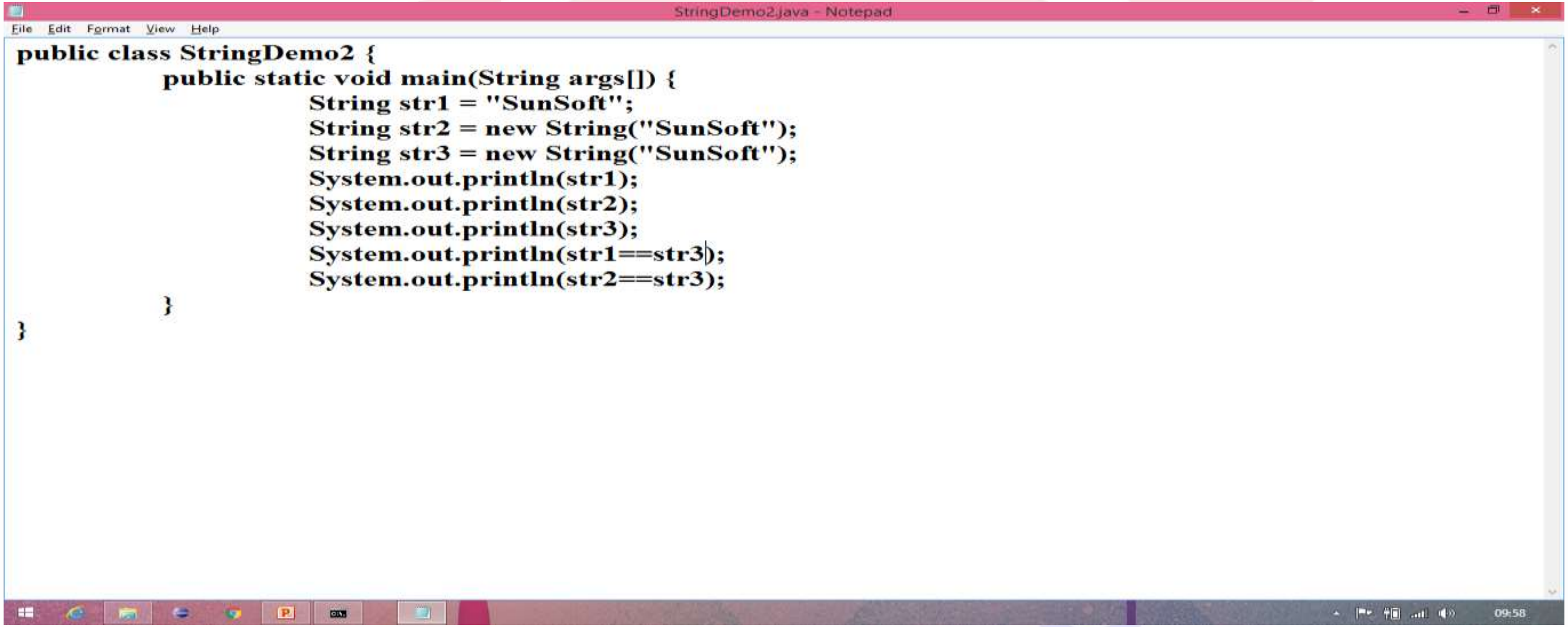
String, StringBuffer, StringBuilder (contd.)

Creating String object with new keyword:

- JVM allocates the memory for the String reference variable.
- JVM verifies the String literal in the String Constant Pool.
- If String literal is not available in the String Constant Pool then it creates new String object inside the String Constant pool.
- If String literal is available in the String Constant Pool then ignores that.
- It creates another new String object outside the constant pool and assigns address of the newly created String object outside the pool to String reference variable.

String, StringBuffer, StringBuilder (contd.)

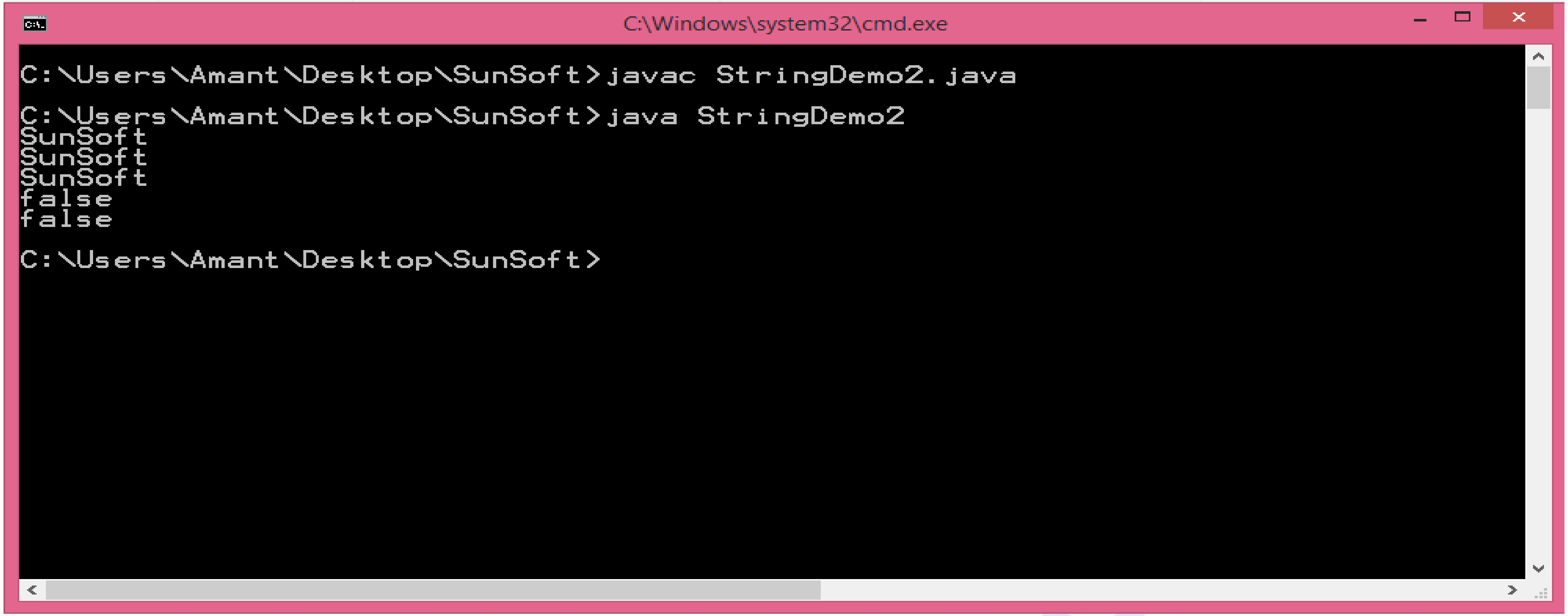
Creating String object with new keyword Demo:



```
public class StringDemo2 {
		public static void main(String args[]) {
				String str1 = "SunSoft";
				String str2 = new String("SunSoft");
				String str3 = new String("SunSoft");
				System.out.println(str1);
				System.out.println(str2);
				System.out.println(str3);
				System.out.println(str1==str3);
				System.out.println(str2==str3);
		}
}
```

String, StringBuffer, StringBuilder (contd.)

Creating String object with new keyword Demo:



```
C:\Users\Amant\Desktop\SunSoft>javac StringDemo2.java

C:\Users\Amant\Desktop\SunSoft>java StringDemo2
SunSoft
SunSoft
SunSoft
false
false

C:\Users\Amant\Desktop\SunSoft>
```

String, StringBuffer, StringBuilder (contd.)

Some useful methods in String class:

- public int length()
- public boolean isEmpty()
- public String concat(String)
- public String toLowerCase()
- public String toUpperCase()
- public String trim()

String, StringBuffer, StringBuilder

StringBuffer:

- ➢ StringBuffer is a final class in java.lang package.
- ➢ StringBuffer is mutable i.e. the content of StringBuffer can be modified after creation.
- ➢ Every StringBuffer object has a capacity associated with it.
- ➢ The capacity of StringBuffer is the number of characters it can hold.
- ➢ The capacity increases automatically as more contents added to it.
- ➢ The methods available in StringBuffer class are synchronized.
- ➢ In StringBuffer class multiple threads cannot access simultaneously.
  - To sole this problem StringBuilder is added from Java 5.

String, StringBuffer, StringBuilder (contd.)

Some useful methods in StringBuffer class:

- ➢ StringBuffer append(X value)
- ➢ int capacity()
- ➢ int length()
- ➢ void setLength(int len)

String, StringBuffer, StringBuilder

StringBuilder:

- StringBuilder is a final class in java.lang package.
- StringBuilder is added from Java 5.
- StringBuilder class functionality is same as StringBuffer only except that the methods available in StringBuilder class are non synchronized.
- In StringBuilder class multiple threads can execute simultaneously.

String, StringBuffer, StringBuilder (contd.)

Some useful methods in StringBuilder class:

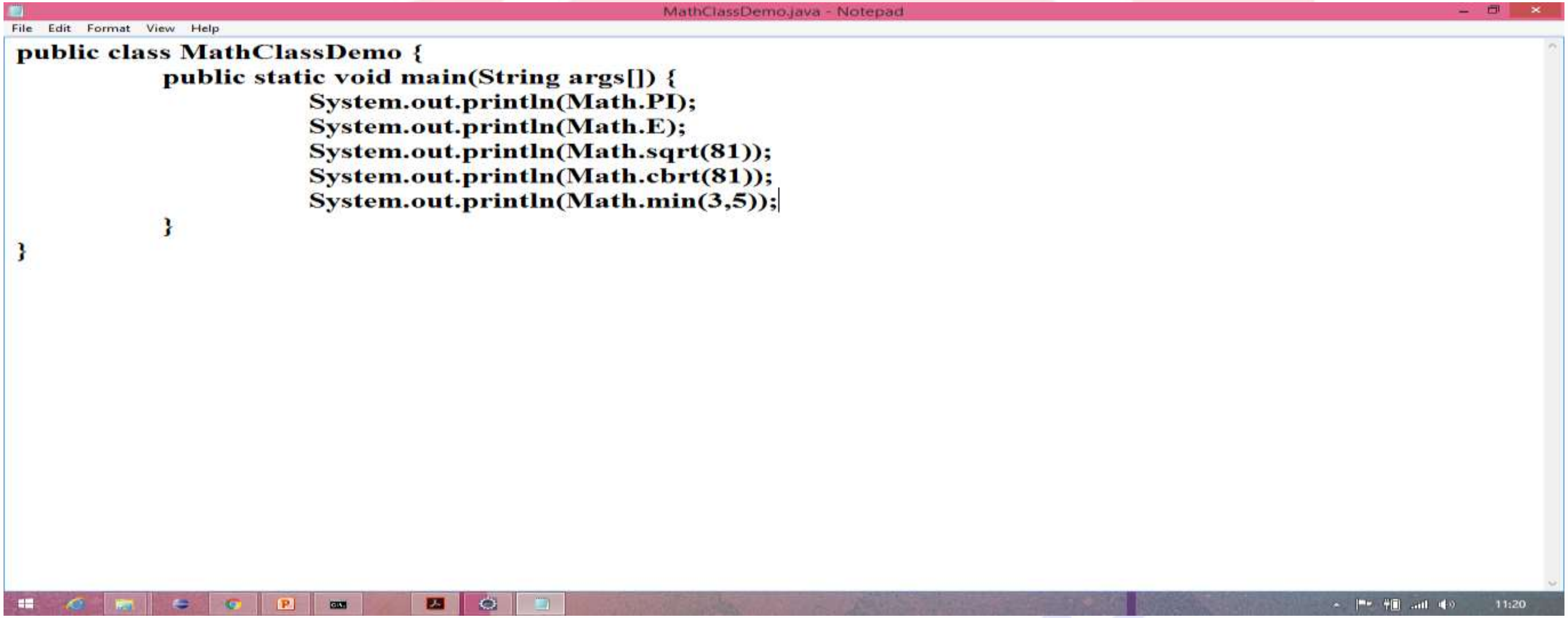
- StringBuffer append(X value)
- int capacity()
- int length()
- void setLength(int len)

Math class

- It is a final class in java.lang package.
- It contains methods for performing basic numeric operations.
- The constructor of Math class is private so you cannot create the object of Math class.
- All the members of Math class is static so it can be accessed by using class name.

Math class

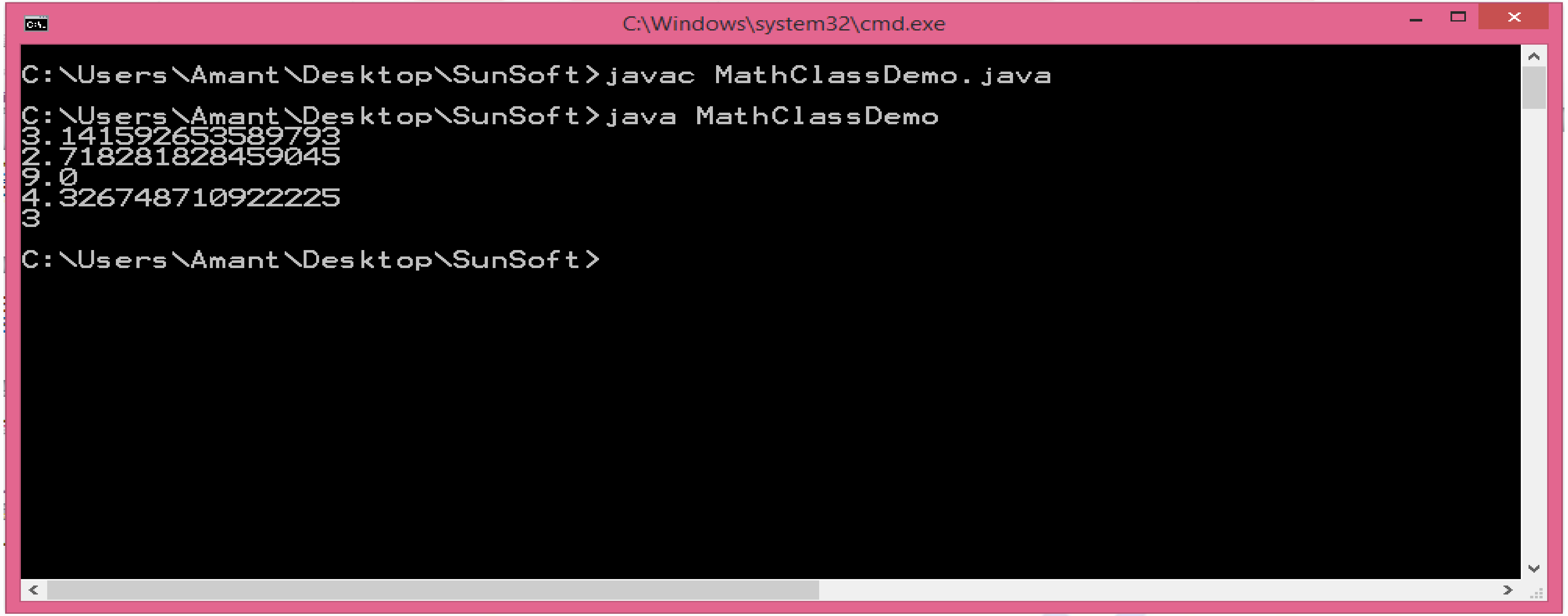
Math class demo:



```
public class MathClassDemo {
		public static void main(String args[]) {
				System.out.println(Math.PI);
				System.out.println(Math.E);
				System.out.println(Math.sqrt(81));
				System.out.println(Math.cbrt(81));
				System.out.println(Math.min(3,5));
		}
}
```

Math class

Math class demo:


C:\Windows\system32\cmd.exe

```
C:\Users\Amant\Desktop\SunSoft>javac MathClassDemo.java

C:\Users\Amant\Desktop\SunSoft>java MathClassDemo
3.141592653589793
2.718281828459045
9.0
4.326748710922225
3

C:\Users\Amant\Desktop\SunSoft>
```

Runtime and Process class

Runtime class:

- Runtime class is present in java.lang package.
- Every java program has the single instance of this class to allow the application to interact with the environment.
- Runtime class is used for memory management and executing additional process.
- You can define the reference of Runtime class but you cannot create the object of Runtime class.
- The methods of the Runtime class are defined as instance, so you need to use some object to invoke the methods.
- Runtime class is implemented using singleton design pattern i.e. only one object of that class will be available per JVM.

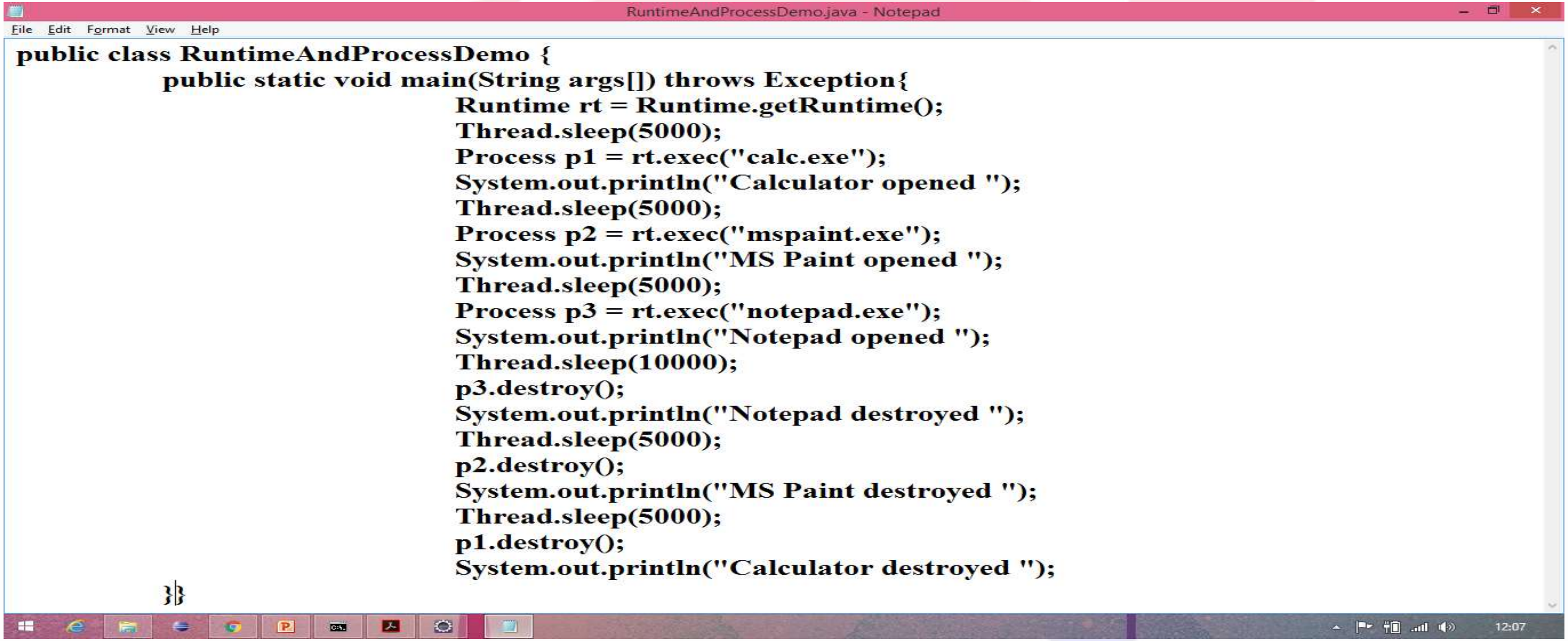
Runtime and Process class

Process class:

- ➢ Process class is present in java.lang package.
- ➢ Process class provides methods for the following operations:
  - Taking input from the process
  - Giving output to the process.
  - Waiting for the process to complete.
  - Destroying the process.

Runtime and Process class

Demo:

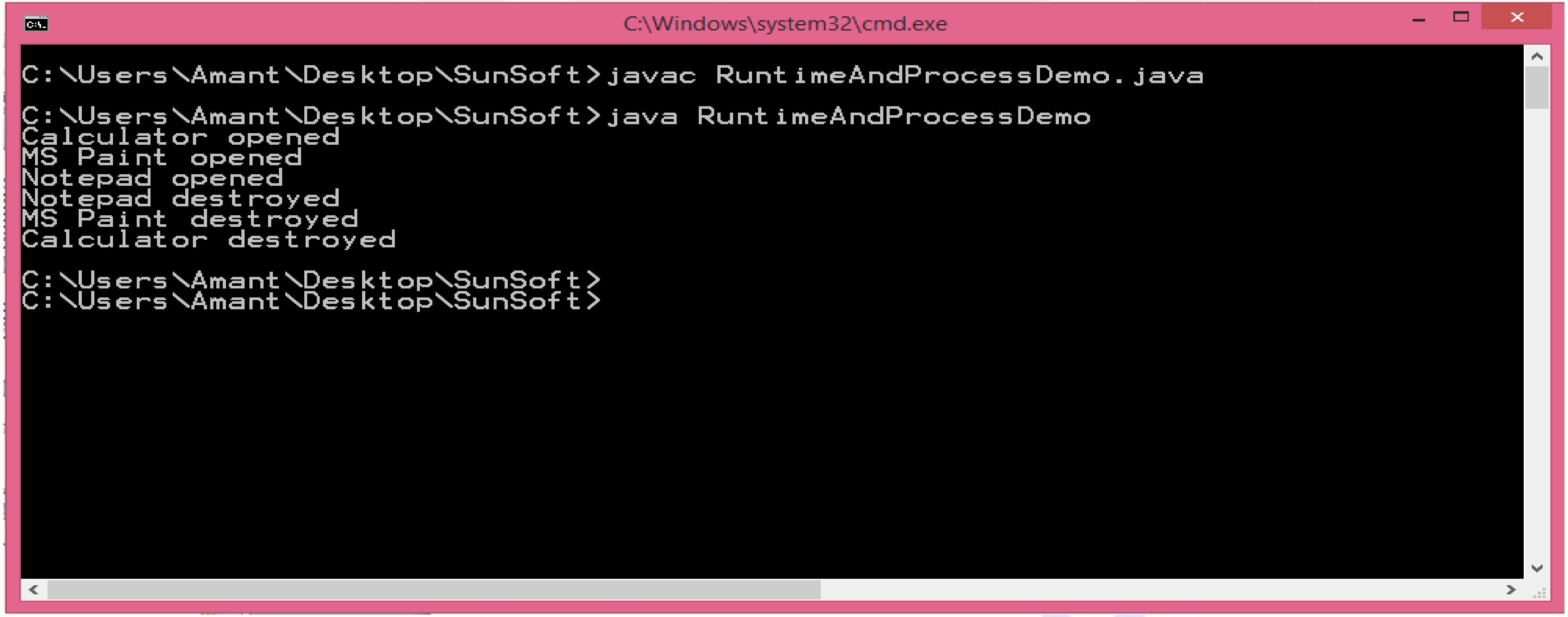


RuntimeAndProcessDemo.java - Notepad
File Edit Format View Help

```
public class RuntimeAndProcessDemo {
		public static void main(String args[]) throws Exception{
				Runtime rt = Runtime.getRuntime();
				Thread.sleep(5000);
				Process p1 = rt.exec("calc.exe");
				System.out.println("Calculator opened ");
				Thread.sleep(5000);
				Process p2 = rt.exec("mspaint.exe");
				System.out.println("MS Paint opened ");
				Thread.sleep(5000);
				Process p3 = rt.exec("notepad.exe");
				System.out.println("Notepad opened ");
				Thread.sleep(10000);
				p3.destroy();
				System.out.println("Notepad destroyed ");
				Thread.sleep(5000);
				p2.destroy();
				System.out.println("MS Paint destroyed ");
				Thread.sleep(5000);
				p1.destroy();
				System.out.println("Calculator destroyed ");
		}}
```

Runtime and Process class

Demo output:



C:\Windows\system32\cmd.exe

```
C:\Users\Amant\Desktop\SunSoft>javac RuntimeAndProcessDemo.java

C:\Users\Amant\Desktop\SunSoft>java RuntimeAndProcessDemo
Calculator opened
MS Paint opened
Notepad opened
Notepad destroyed
MS Paint destroyed
Calculator destroyed

C:\Users\Amant\Desktop\SunSoft>
C:\Users\Amant\Desktop\SunSoft>
```

Util Package Classes

- Following are the useful classes of util package
    a. Date
    b. Stack
    c. StringTokenizer
    d. BitSet
    e. Scanner etc.

Date class

- Date Is a utility class present in util package used to handle all date operation
- You can import util package like java.util.Date;

Ex

```
import java.util.Date;

public class Dat {

public static void main(String[] args) {
Date d=new Date();
System.out.println(d);
}
}
```

Stack class

- The **java.util.Stack** class represents a last-in-first-out (LIFO) stack of objects.
- It is synchronized class
- In this class, the last element inserted is accessed first
- It is a legacy class not in use

StringTokenizer

- ➢It is class present is java.util.StringTokenizer
- ➢It is used to split a String into diffrent tokens by defined delimiter
- ➢You can defined more token in StringTokenizer to split String into different Token

Ex :

```
import java.util.*;

public class StringTokenizerDemo {

public static void main(String[] args) {
String str="raja,smaeer.raheem;sham:";
StringTokenizer s=new StringTokenizer(g,",.:;");
while(s.hasMoreTokens()){
System.out.println(s.nextToken());
}
}
}
```

Out put :

```
raja
smaeer
raheem
sham
```

BitSet class

- The BitSet class creates a special type of array that holds bit values
- The BitSet array can increase in size as needed
- This is a legacy class but it has been completely re-engineered in Java 2, version 1.4.

Scanner class

- Scanner class is a final class in java.util.Scanner
- Scanner class is used to read the data from input devices

Demo :

```
Import java.util.Scanner;
class A{
public static void main(String[] args){
System.out.println(“enter value”);
Scanner in =new Scanner();
String s=in.nextLine();
System.out.println(s);
}}
```

Collection

- Collection is a interface available in java.util package
- Collection is used to represent group of Object as a single entity
- Collections having so many advantage over arrays so that collection is using widely

Differences between array and Collection

Disadvantages of array

- Array hold homogenous data type

  ex : int a[]=new int[5];

- Array size is fixed means after initializing an array you cannot change the size of an array
- Size of an array has to declare while initializing an array
- Performance will be low when operation is insertion and deletion

Advantages Collection

- Collection hold heterogeneous element

ex : ArrayList al=new ArrayList();

- Collection are growable in nature means size is not fixe
- no need to declare any size while initializing an Collection

  ex: LinkedList l=new LinkedList();

- The root class of Collection is iterable interface

Some of the common method in Collection

1.add(Obj) : to add element

2.addAll(Collection c) : to add Collection

3.remove(obj) : to remove object

4.removeAll(Collection c) : remove Collection

5.retainAll(Collection c) : remove remaining element except these collection

6.size() : return size of the collection

7.clear() : remove all element from collection

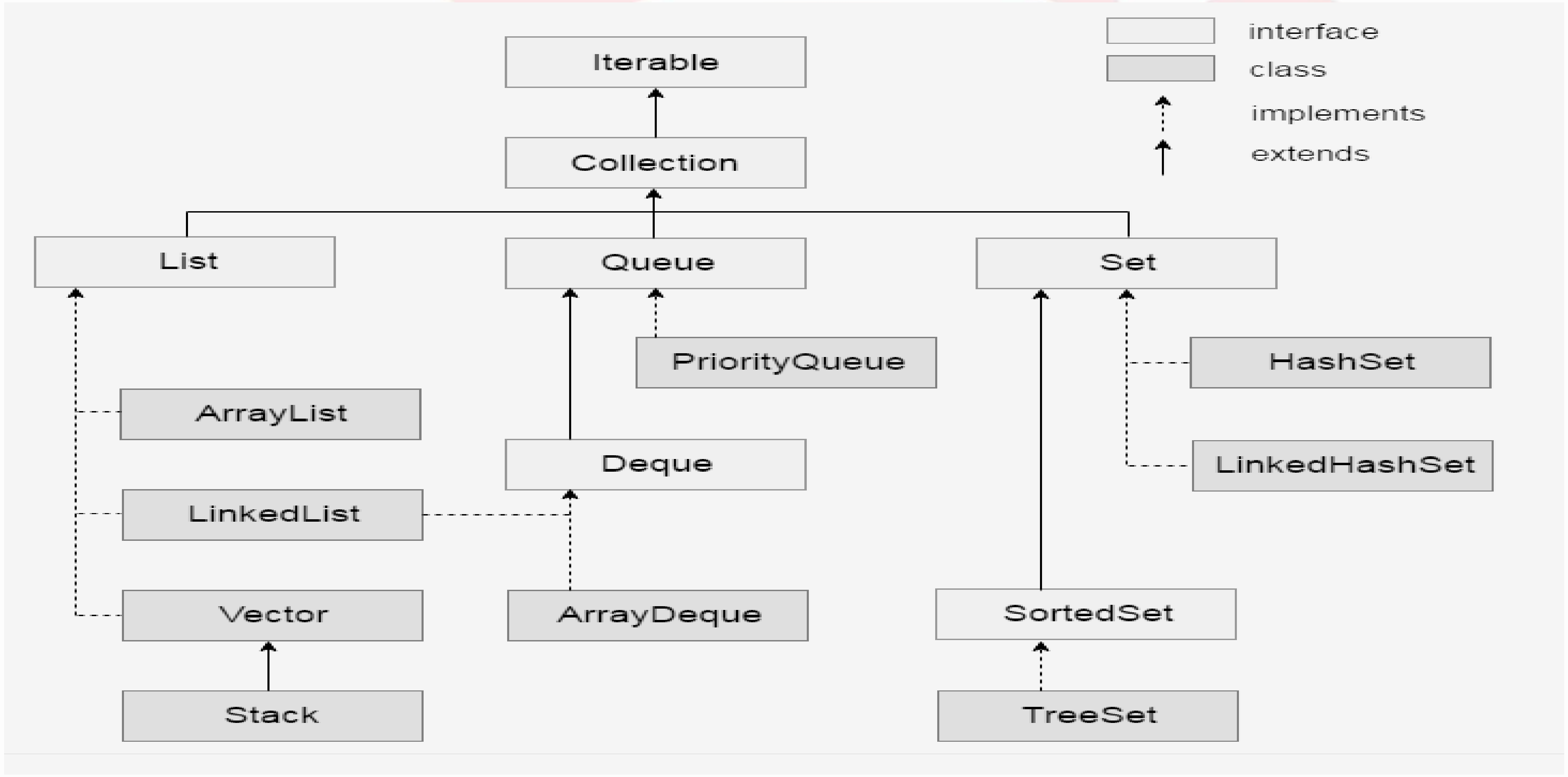
8.contains(Obj) : return true if object is present

9.containsAll(Collection c) : return true if collection is present

10.isEmpty() : return true if no element in collection

11.toArray[] : convert Collection to Object Array

Classification of Collection



Classification of Collection

- ❖ Iterable

  - ❑ Collection

    I. List

    II. Queue

    III. Set

i. List

1.ArrayList

2.LinkedList

3.vector

stack

1.ArrayList

- It store element in indexing notation means in continuous memory location
- It allow duplicate element
- It store element in user added order
- Performance is good when operation is retrieval operation
- Performance is poor when operation is insert ,delete (more shifting of element)
- It is non Synchronized

Syntax to create ArrayList and methods

ArrayList al=new ArrayList();

Methods:

- It implements Collection so Collection all method will be access

1.add(int a, Obj) : add element on specified index

2.remove(int a) : remove element from specified index

3.lastIndexOf(obj) : returns the last occurrence index of specified element

4.addAll(int index, Collection) : adding Collection into collection from specified index

5. **indexOf(Object o) :** return index of specified Object

- ArrayList implements random-access interface so retrieval operation is fast

Difference between Array and ArrayList

Array	ArrayList
• Array is static (fixed size)	• it is dynamic (growable size)
• Need to specify size while creating array object	• No need to specify size
• Need length property to get size of array	• Need size() to get length of collection
• Need to specify type of element or data type	• No need to specify type of elements or data type

LinkedList

- ➢It store the element in doubly linked list
- ➢It allows duplicate element
- ➢it is non Synchronize
- ➢The performance is good when Operation is insert and deletion
- ➢The performance is poor when Operation is Retrieval
- ➢It does not implement randomAcess interface

Storing of element

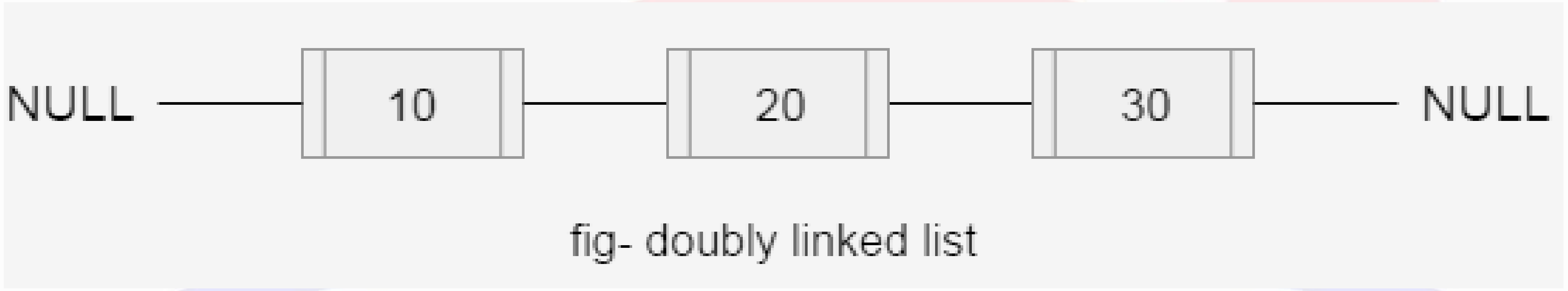


fig- doubly linked list

Syntax to create LinkedList
LinkedList al=new LinkedList();

Methods in LinkedList

1.add(int index, Obj) : add element in specified index

2.addFirst(Obj) : add element on beginning index

3.addLast(Obj : add Object on last

4. Object getFirst() : return first element

5. Object getLast() : return last element

6. int lastIndexOf(Object o) : return index of last element

Differences between ArrayList and LinkedList

ArrayList	LinkedList
Stores element in indexing notation	It stores elements in doubly linked list
Performance is fast when operation is retrieval	Performance is poor when operation is retrieval
Performance is poor when operation is insert delete	Performance is fast when operation is insert delete
Occupies less memory	Occupies more memory as links contain address of elements

Vector and stack

- Vector is a legacy class
- All methods are synchronized
- Stores element in index
- It allow duplicate element
- growable in nature
- null insertion is allowed
- Stack is not in use we other class which better then stack like HashSet

Differences between vector and ArrayList

Vector	ArrayList
• Methods are Synchronized	• Methods are non synchronized
• You can use any cursor to access elements	• You cant use Enumeration cursor to access elements
• It is thread safe means multiple thread will not access this object	• It is not thread safe means multiple thread can access this object

Enumeration interface

- ➢It is interface available in java.util package
- ➢It is an cursor to access the elements form legacy classes one by one

Methods

1.boolean hasMoreElements() :checks whether next element available or not

2.Object nextElement() : moves the pointer and returns elements from collection

Iterator interface

- This is a cursor used to access elements from collection one by one
- Invoke iterator() on collection to get instance of iterator ex: iterator i=col.iterator()

Methods

1.hasNext() : checks whether next element available or not

2.next() : moves the pointer and returns elements from collection

3.remove() : remove the current element from the collection

Differences between Enumeration and iterator

Enumeration

- It is legacy interface
- It is used to access elements from legacy classes(vector, stack,

  HashTable)

You can not remove elements using this

Iterator

- It is collection framework interface
- It is universal cursor can be used to access elements from all class
- You can remove elements

  by Iterator

ListIterator

- ➢It extends iterator interface
- ➢Used to access the elements
- ➢Access the elements both forward and reverse direction
- ➢Add, remove, replace operation can be done
- ➢Access elements from specified index

Differences between ListIterator and Iterator

ListIterator	Iterator
• It can be used to access elements from any Collection class	• It is used to access elements from list and its subclasses
• Can access elements Both forward and reverse direction	• Can access elements in forward direction only
• You can add, remove replace elements	• You can only remove the elements
• With this you can access elements from specified index	• With this you can access elements from beginning only

Queue

- ➢ Queue is an interface
- ➢PriorityQueue is famous class of this interface

Syntax to create Queue

PriorityQueue pq=new PriorityQueue();

Methods in Queue

1. offer(object): used to add Object
2. Object poll() : used to retrieve and remove head of the Queue
3. Object peek() : used to retrieve and not remove head of the element
4. Object element() : used to retrieve element but not remove head

Set Interface

- ➢It is not allowed duplicate elements
- ➢It uses hash code and equals() to add Object
- ➢Methods are same as Collection interface

classes in Set

1.HashSet

2.LinkedHashSet

3.TreeSet

1.HashSet

- It store elements in unordered
- It is fast for searching and retrieval operation
- Duplicate are not allowed

2.LinkedHashSet

- It is subclass of HashSet
- It store elements in user added order

3.TreeSet

- It stores elements in sorted order
- compareTo() and Comparable interface is used to sort the elements
- It allows to store similar type elements only
- Null value insertion is not allowed

syntax to create TreeSet

```
        TreeSet t=new TreeSet();
```

Differences between List and Set

List	Set
It can hold duplicate elements	It cant hold duplicate elements
You can add elements to the required position	You can not add elements to the required position
You can remove elements from the required position	You can not remove elements from the required position
It does not store elements in sorted order	It store elements in sorted order

Map Interface

- It is Collection frame API but does not extends Collection interface
- Map is an Object that amps or store key and value
- The key Is always unique and value can be same
- One value one key pair is known as one map entry

Method s in Map

1.Int size() : return number of entry of map

2.isEmpty() : return true if map is Empty

3.containskey(Obj) : return true if key is available

4.containsValue(Obj) : return true if available

5.get(key) : return Object of specified Key

6.put(k ,v) :add map entry

7.clear() : remove all elements

8.PutAll() : add map into map

Classes of Map

1.HashMap

- LinkedHashMap

2.TreeMap

3.HashTable

1.HashMap

- ➢It is fast for searching and inserting elements
- ➢It store elements in unordered

- **LinkedHashMap**

➢It is a subclass of HashMap

➢Elements Store in user added order

3.TreeMap

➢It store elements in sorted order

➢compareTo() and Comparable Interface I s used to sort elements

➢It allow similar type of elements only

➢Null value cannot be stored

Generics

- Collection can hold any data type so while performing some operation on data there mat chances of Exception
- Generics is used to restrict the user to provide homogenous data type
- Diamond operator is use for that

  ex: ArrayList<String>=new <String>ArrayList();

Collections class

- Collections is utility class present in util package

- Collections class utility methods that can be used to manipulate the Collection element like sorting ,shuffling, searching etc.

Demo :

```
import java.util.*;

class col{
public static void main(String[] args){
ArrayList l=new ArrayList();
l.add("raju");
l.add("raam");
l.add("raheem");
System.out.println(l);
Collections.shuffle(l);
System.out.println(l);
}
}
```

static import

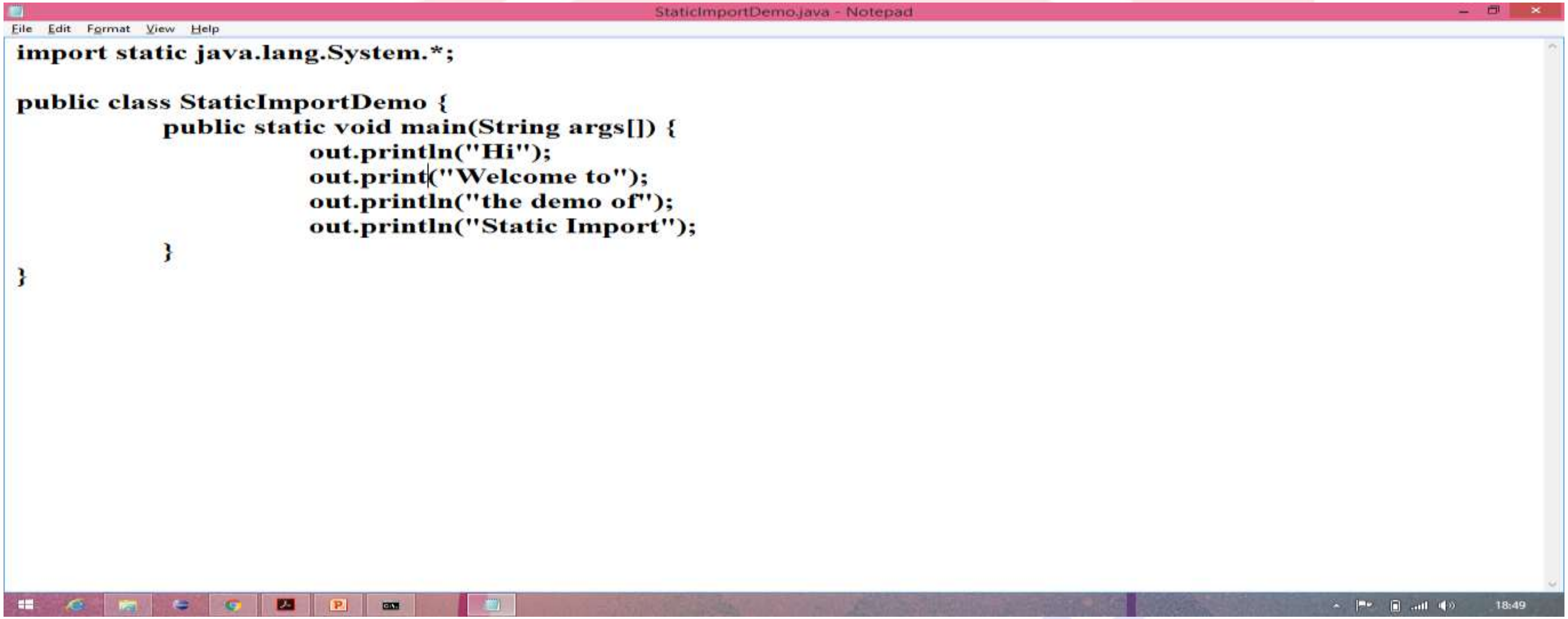
- static import allows programmer to access any static member of class directly.
- static import was introduced from Java 5.
- static import shorten the code.

Syntax:

```
import static java.lang.Math.*;
import static java.lang.System.*;
```

Static import (contd.)

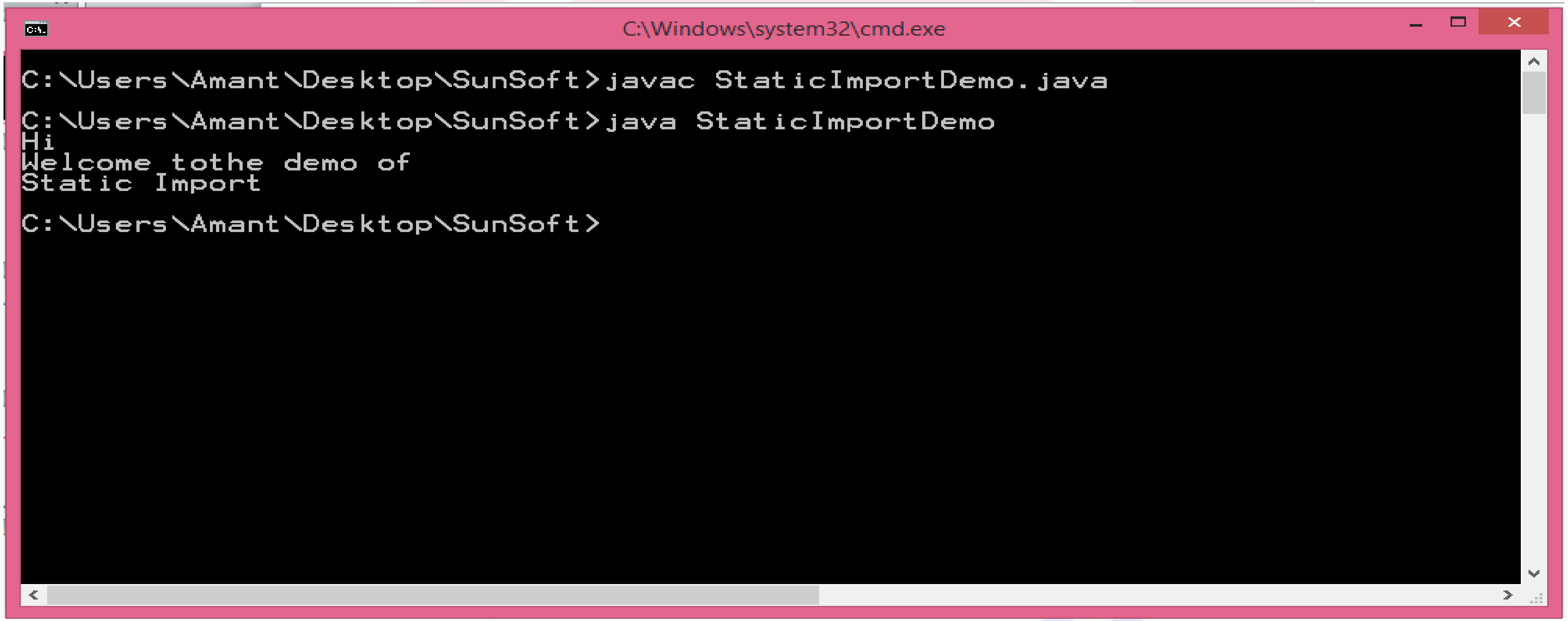
Demo:



```
import static java.lang.System.*;

public class StaticImportDemo {
		public static void main(String args[]) {
				out.println("Hi");
				out.print("Welcome to");
				out.println("the demo of");
				out.println("Static Import");
		}
}
```

Static import (contd.)

Demo:



```
C:\Users\Amant\Desktop\SunSoft>javac StaticImportDemo.java

C:\Users\Amant\Desktop\SunSoft>java StaticImportDemo
Hi
Welcome tothe demo of
Static Import

C:\Users\Amant\Desktop\SunSoft>
```

Enums

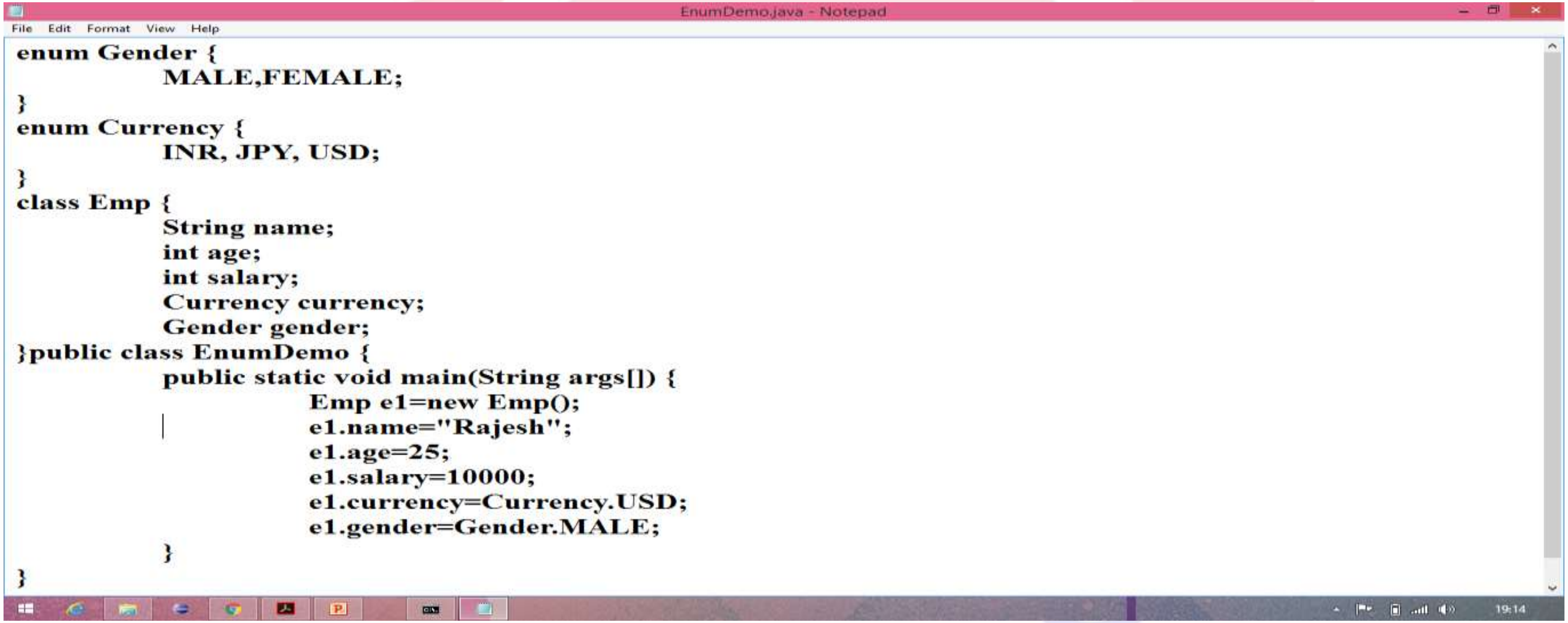
Enums are user defined data types in which programmer fix some value.

- Enums were introduced from java 5.
- Enums can also be called as special type of class.
- Basically enums are used to fix a set of constrains.
- Enums can have constructors, methods and fields.
- Enums internally extends Enum class, so enums cannot extend any class but any number of interfaces can be implemented.
- Whenever compiler creates enum it internally adds values() method.

  The values() methods returns an array containing all the values of the enum.

Enums (contd.)

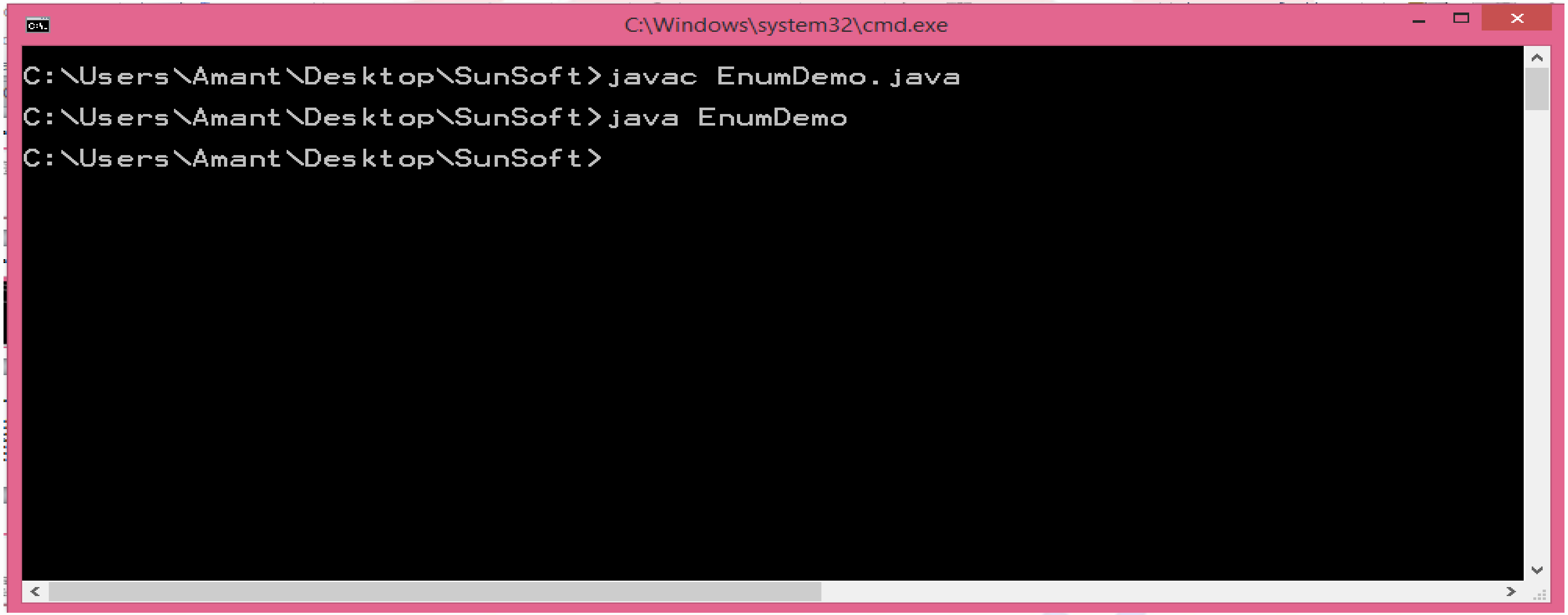
Demo:



```
enum Gender {
		MALE,FEMALE;
}
enum Currency {
		INR, JPY, USD;
}
class Emp {
		String name;
		int age;
		int salary;
		Currency currency;
		Gender gender;
}public class EnumDemo {
		public static void main(String args[]) {
				Emp e1=new Emp();
				e1.name="Rajesh";
				e1.age=25;
				e1.salary=10000;
				e1.currency=Currency.USD;
				e1.gender=Gender.MALE;
		}
}
```

Enums (contd.)

Demo:



```
C:\Users\Amant\Desktop\SunSoft>javac EnumDemo.java

C:\Users\Amant\Desktop\SunSoft>java EnumDemo

C:\Users\Amant\Desktop\SunSoft>
```

Var-args

- ➢ Var-args stands for **Variable Arguments**.
- ➢ Var-args is used to take ‘n’ number of inputs from the user.
  - Where ‘n’ can be any number of arguments.

Note:

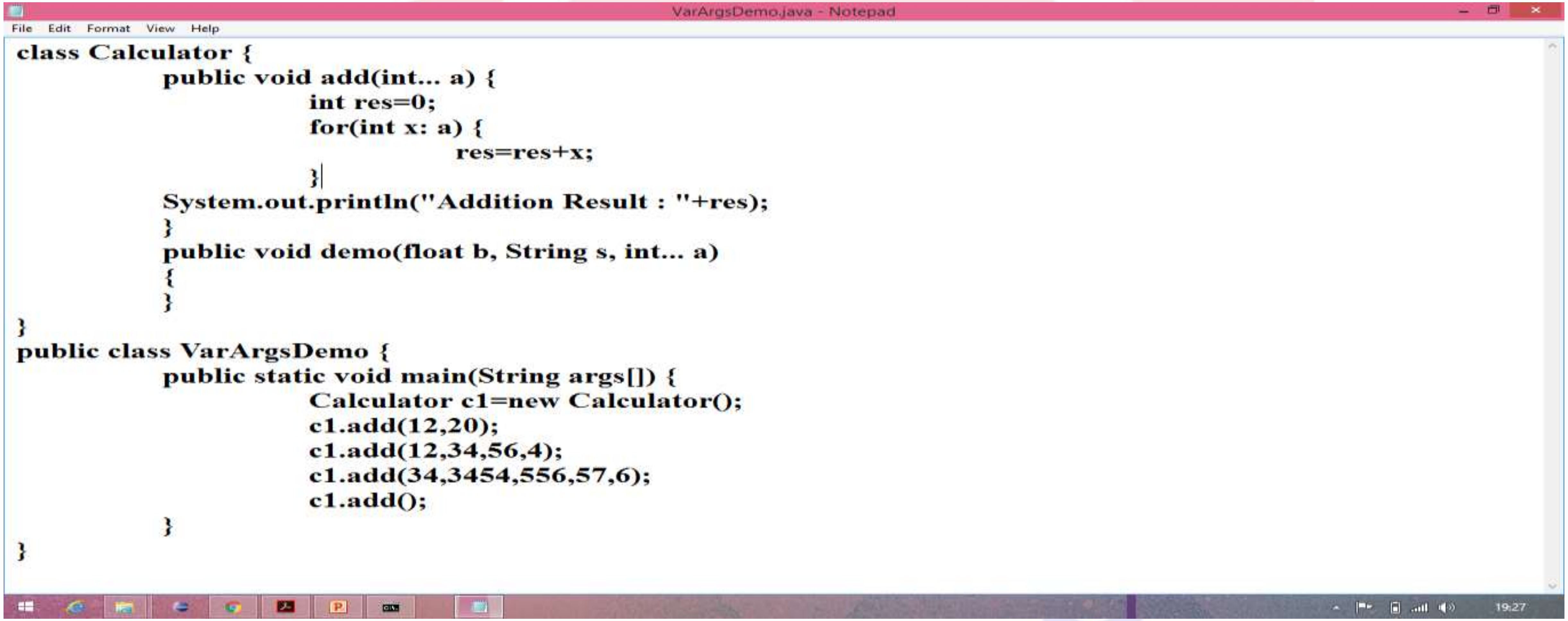
- ➢ In one method only one var-args is allowed.
- ➢ Only last argument in a method can be var-args.

Syntax:

public void method(String s, int… a)	**valid**
public void method(int… a, String s)	**invalid**

Var-args (contd.)

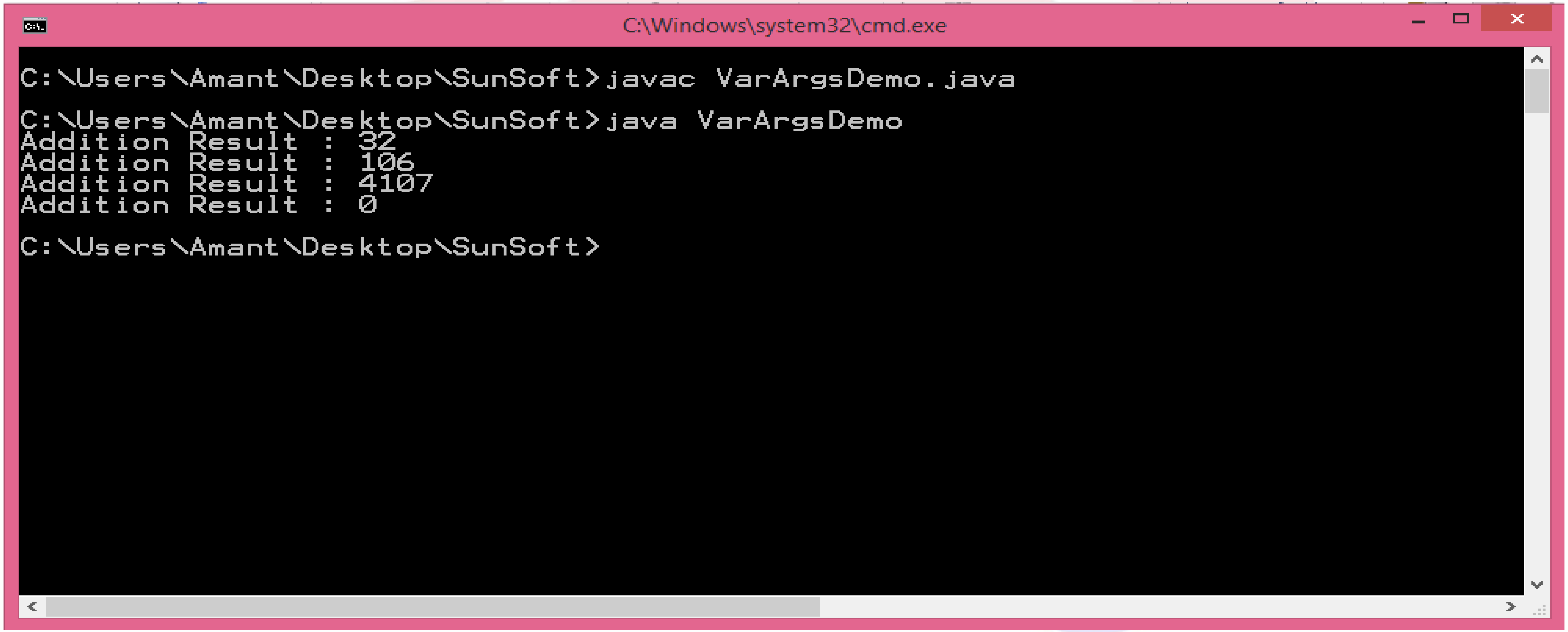
- **Demo:**



```
class Calculator {
                public void add(int... a) {
                                int res=0;
                                for(int x: a) {
                                                res=res+x;
                                }
                System.out.println("Addition Result : "+res);
                }
                public void demo(float b, String s, int... a)
                {
                }
}
public class VarArgsDemo {
                public static void main(String args[]) {
                                Calculator c1=new Calculator();
                                c1.add(12,20);
                                c1.add(12,34,56,4);
                                c1.add(34,3454,556,57,6);
                                c1.add();
                }
}
```

Var-args (contd.)

- **Demo:**



C:\Windows\system32\cmd.exe

```
C:\Users\Amant\Desktop\SunSoft>javac VarArgsDemo.java

C:\Users\Amant\Desktop\SunSoft>java VarArgsDemo
Addition Result : 32
Addition Result : 106
Addition Result : 4107
Addition Result : 0

C:\Users\Amant\Desktop\SunSoft>
```