OOP: Fundamentals

Agenda:

Static Properties: Variables & Methods
Varargs: Variable-Length Arguments
Using Command-Line Arguments
Introducing Access Control

Md. Mamun Hossain
B.Sc. (Engg.) & M.Sc. (Thesis) in CSE , SUST
Assistant Professor, Dept. of CSE, BAUST

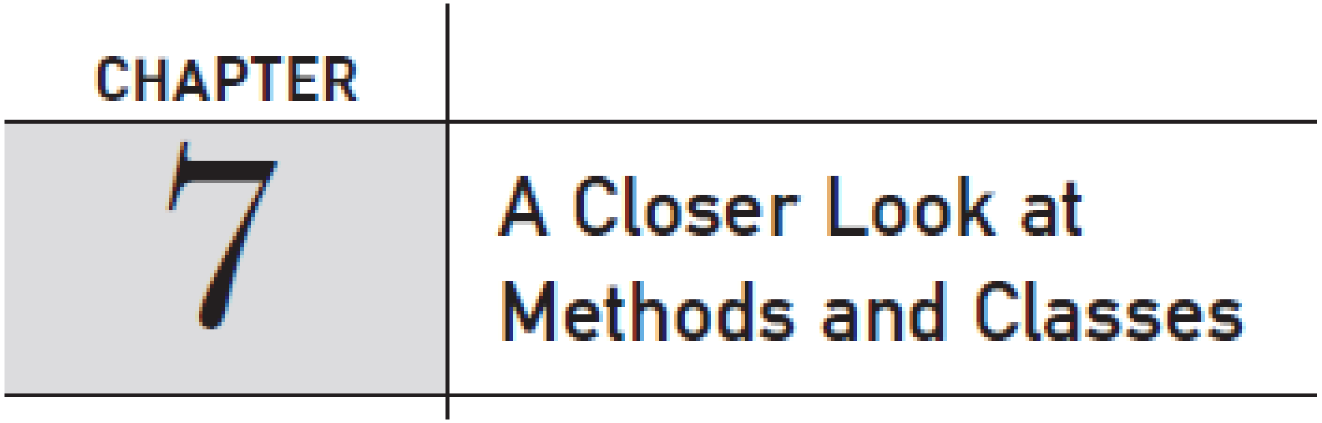
OOP Fundamentals: Core Concepts

Keywords:

- ✓ **Class, Object: Declaration and Assignment**
- ✓ **Method: Simple& Parameterized,**
- ✓ **Constructor: Default& Parameterized**
- ✓ **The dot(.) operator, initialization of variable: Three ways**
- ✓ ***this* keywords, Garbage Collection, Finalize method()**
- ✓ **Modifiers: Access & Non-access Modifiers**
- ✓ **Java Static Properties: Variables and Methods**
- ✓ **Var-arg and Command line arg**
- ✓ **Inheritance: Multiple and Multilevel**
- ✓ **Polymorphism: Overloading & Overriding**
- ✓ **Execution Sequence of Constructors,**
- ✓ **DMD-Dynamic Method Dispatch, Super and Final.**

CHAPTER -7

A Closer Look at Method and Classes



CHAPTER

7

A Closer Look at Methods and Classes

Keywords:

Introducing Access Control

Understanding static

Introducing final

Varargs: Variable-Length Arguments

Using Command-Line Arguments

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Modifiers in Java

- There are two types of modifiers in java:
  - **Access modifiers** and **Non-access modifiers**.
- **Access Modifiers : controls the access level**
  - The access modifiers in java specifies ***accessibility*** (***scope***) of a ***data member, method, constructor or clas***s.
  - There are 4 types of access modifiers in java : private, default, Protected and Public
- **Non-access modifiers :do not control access level, but provides other functionality**
  - There are many non-access modifiers such as ***static***, ***final***, ***abstract***, synchronized, native, volatile, transient etc.

Here, we will learn access modifiers.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers

- **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers

- Java's access modifiers are **public**, **private**, and **protected**. Java also defines a ***default*** access level.

  protected applies only when inheritance is involved.

public and **private**.

  - ✓ When a member of a class is modified by **public**, then that member can be accessed by any other code.
  - ✓ When a member of a class is specified as **private**, then that member can only be accessed by other members of its class.

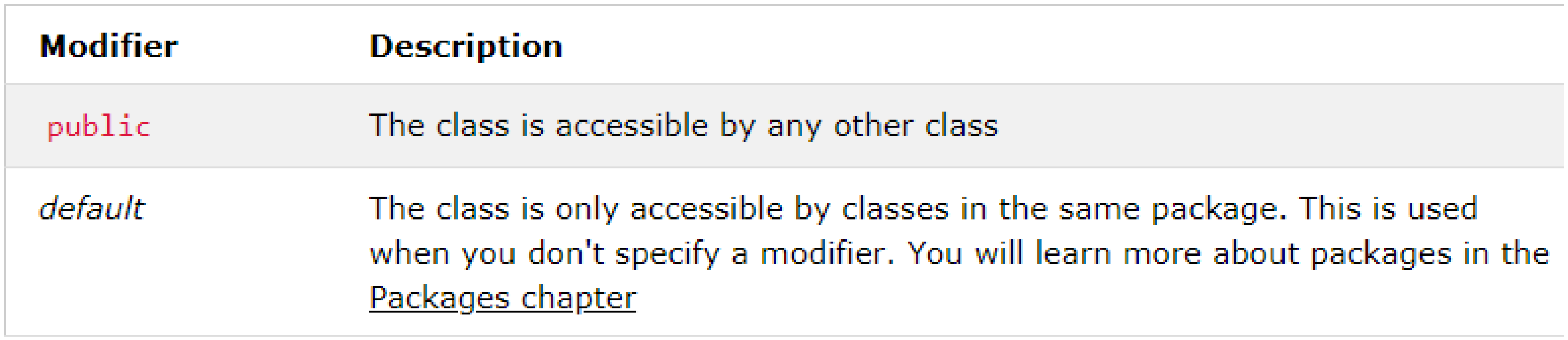
- Now you can understand why **main( )** has always been preceded by the **public** modifier. It is called by code that is outside the program—that is, by the Java run-time system.
- When no access modifier is used, then by default the member of a class is public within its own package, but cannot be accessed outside of its package.

(Packages are discussed in the following chapter. 9)

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers : Class

- For classes, you can use either ***public*** or ***default***:



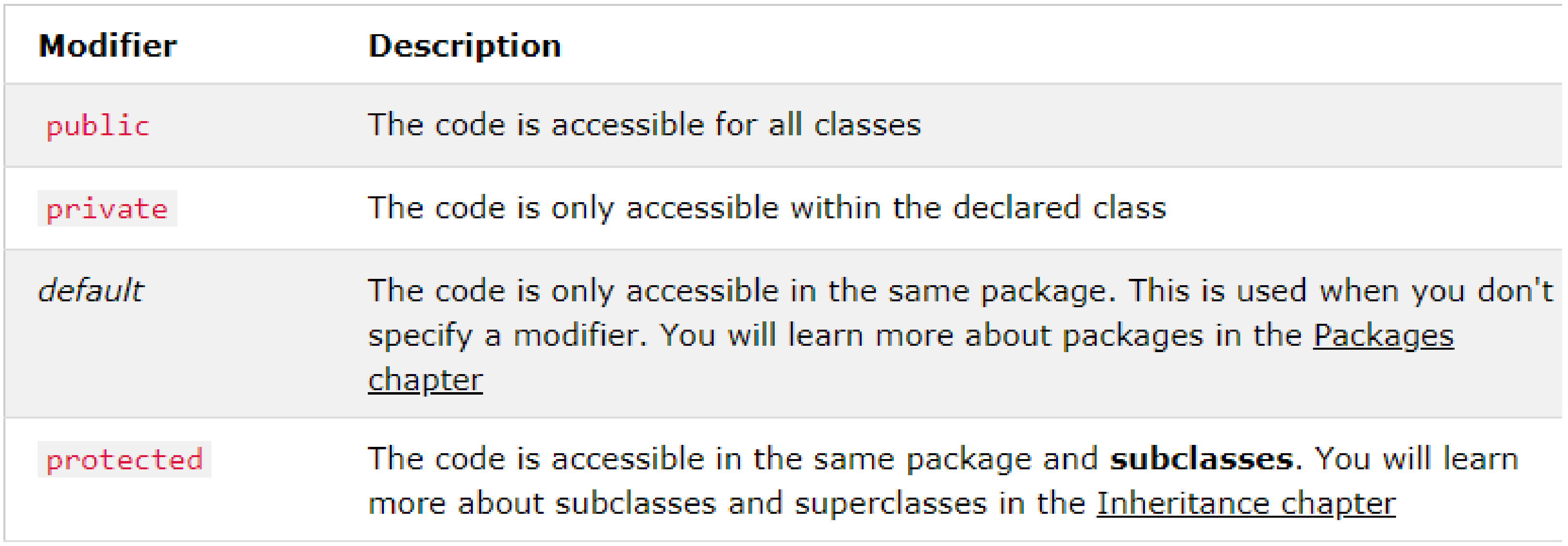
Modifier	**Description**
public	The class is accessible by any other class
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter

A class cannot be private or protected except nested class

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers : Attributes & Methods

- For attributes, methods and constructors, you can use the one of the following:



Modifier	Description
`public`	The code is accessible for all classes
`private`	The code is only accessible within the declared class
default	The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter
`protected`	The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter

A class cannot be private or protected except nested class

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

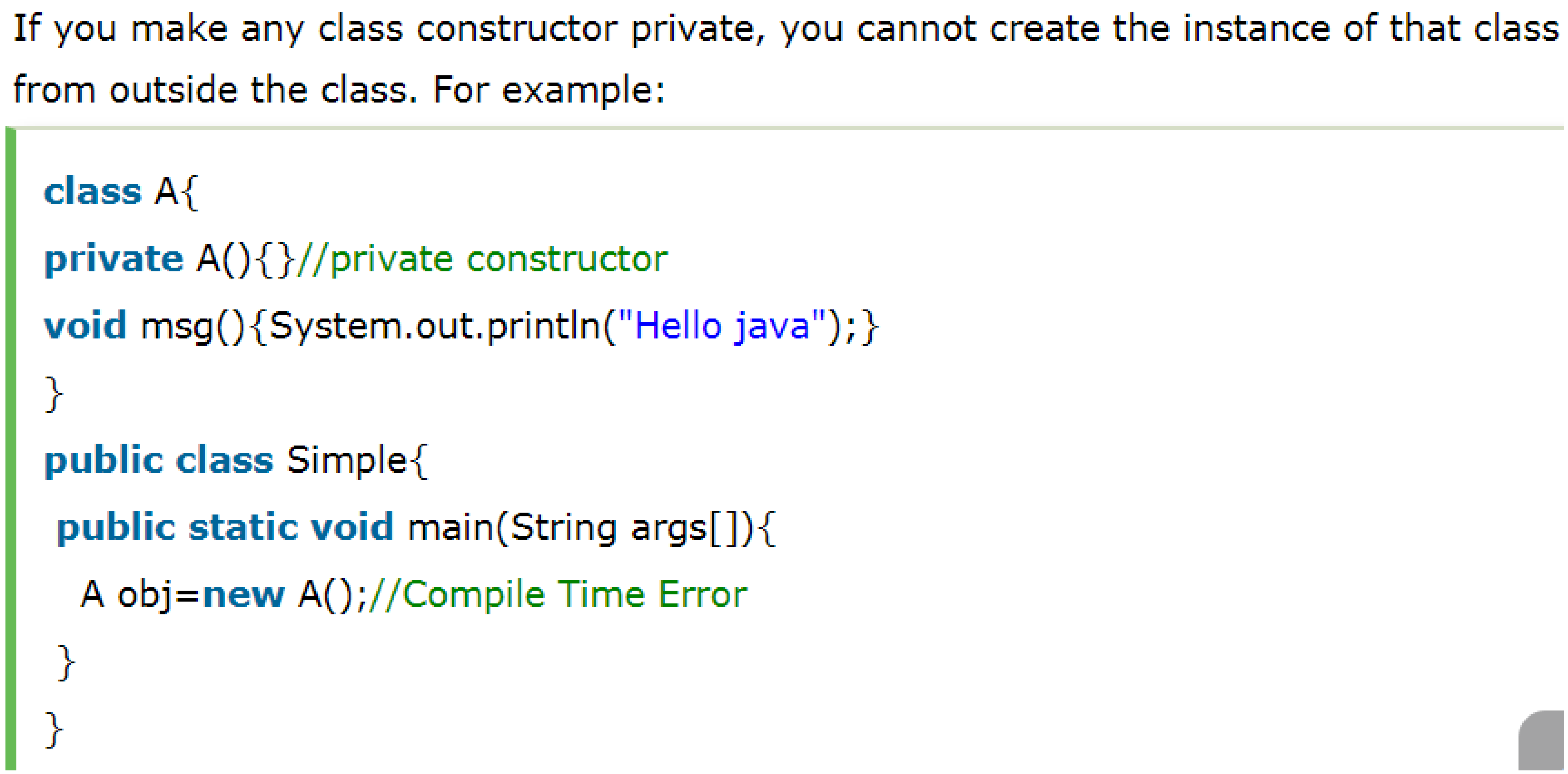
Access modifiers: Private

- The private access modifier is accessible only within class

```java
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers: Private Constructor



If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

A class cannot be private or protected except nested class

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers: Default

- If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package.

```java
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
  public static void main(String args[]){
   A obj = new A();/
   obj.msg();
  }
}
```

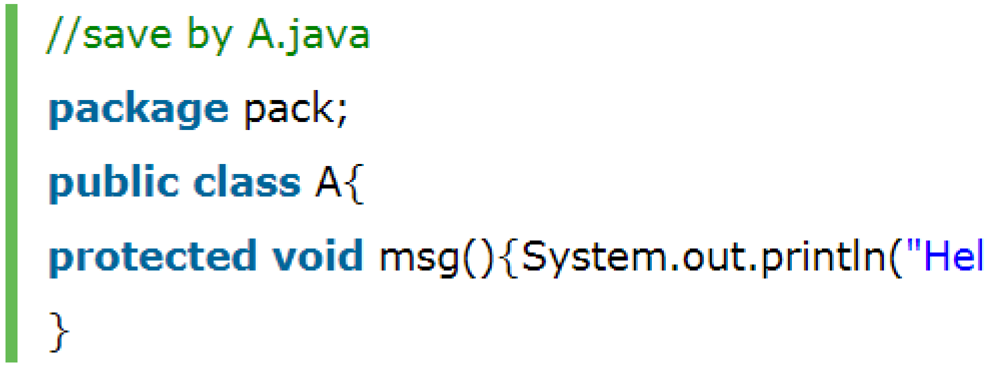
```java
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

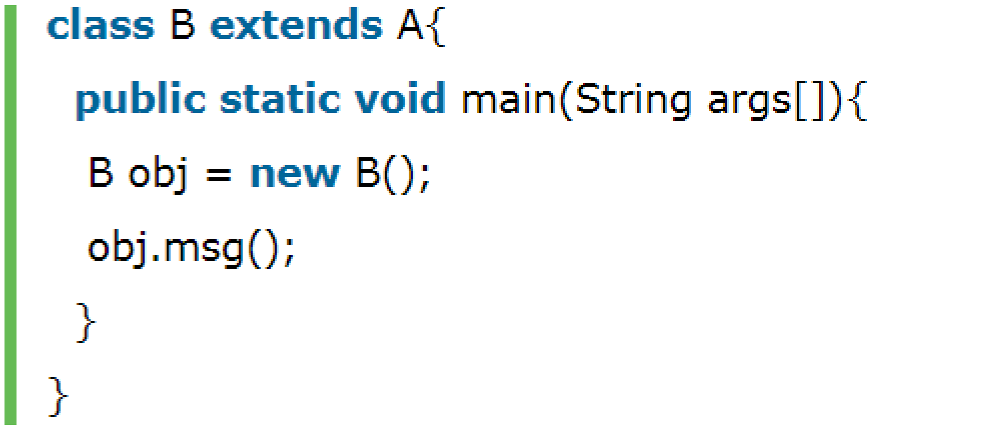
Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers: Protected

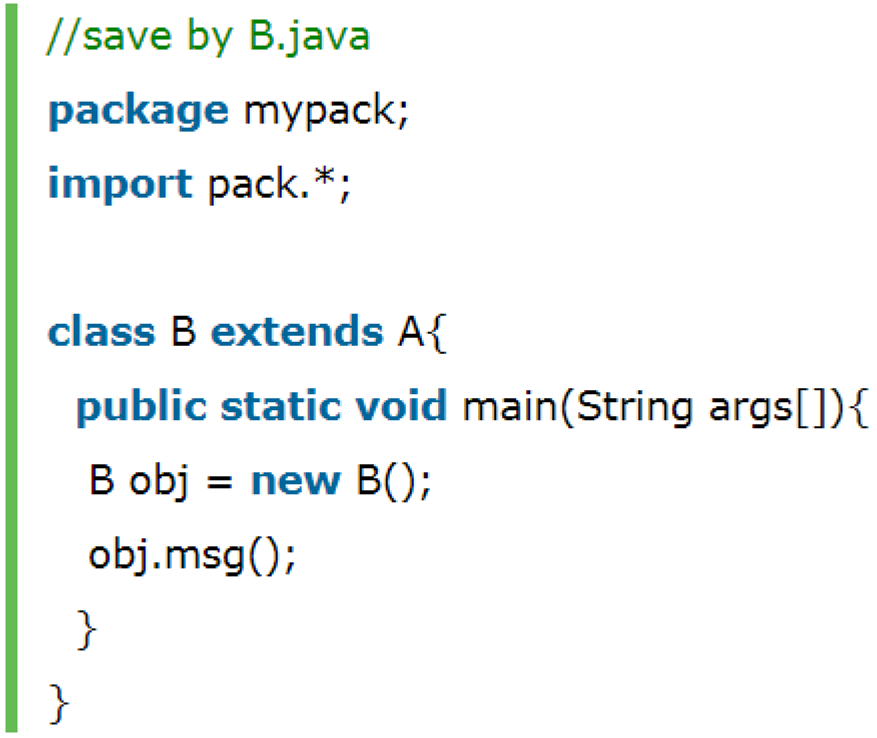
The protected access modifier is accessible within package and outside the package but through inheritance only.



```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hel
}
```



```
class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```



```
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers: Public

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
package pack;
public class A{
public void msg(){
 System.out.println("Hello");
  }
}
```

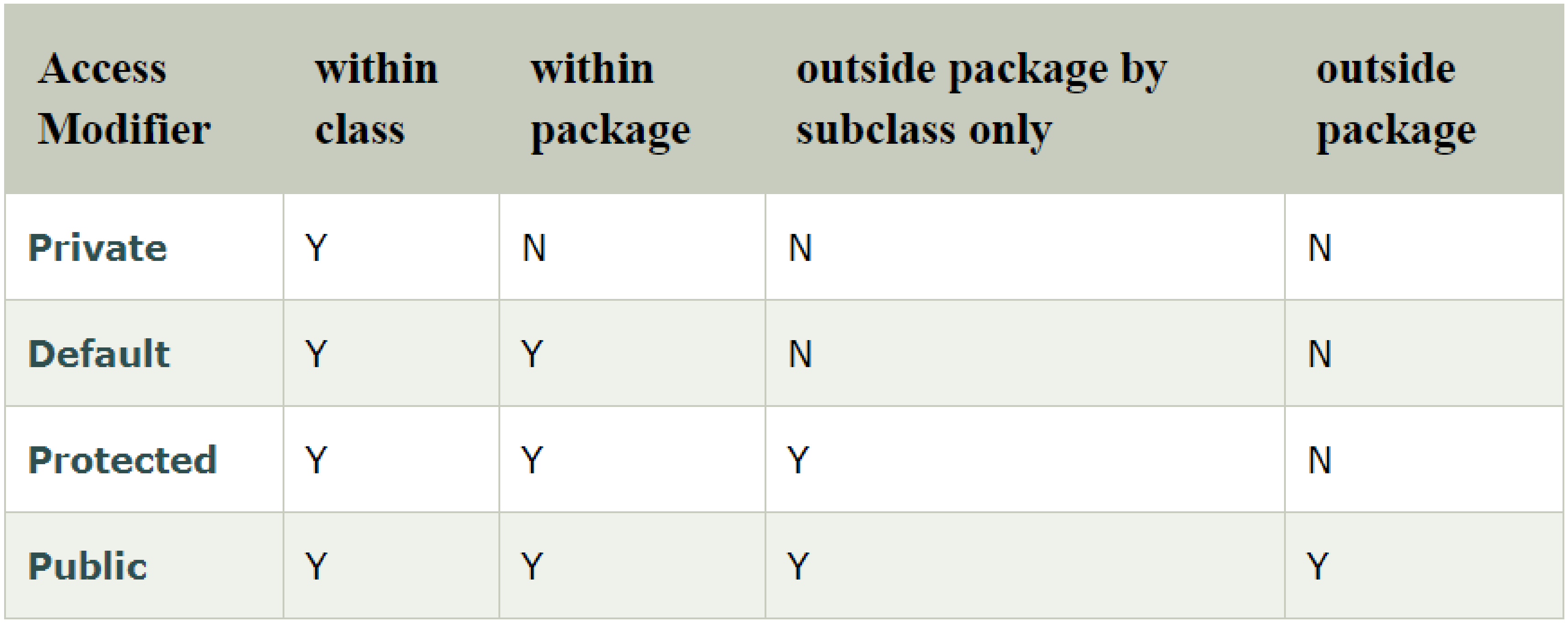
```
class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

```
//save by B.java


package mypack;
import pack.*;


class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers: Access Table

Let's understand the access modifiers by a simple table.

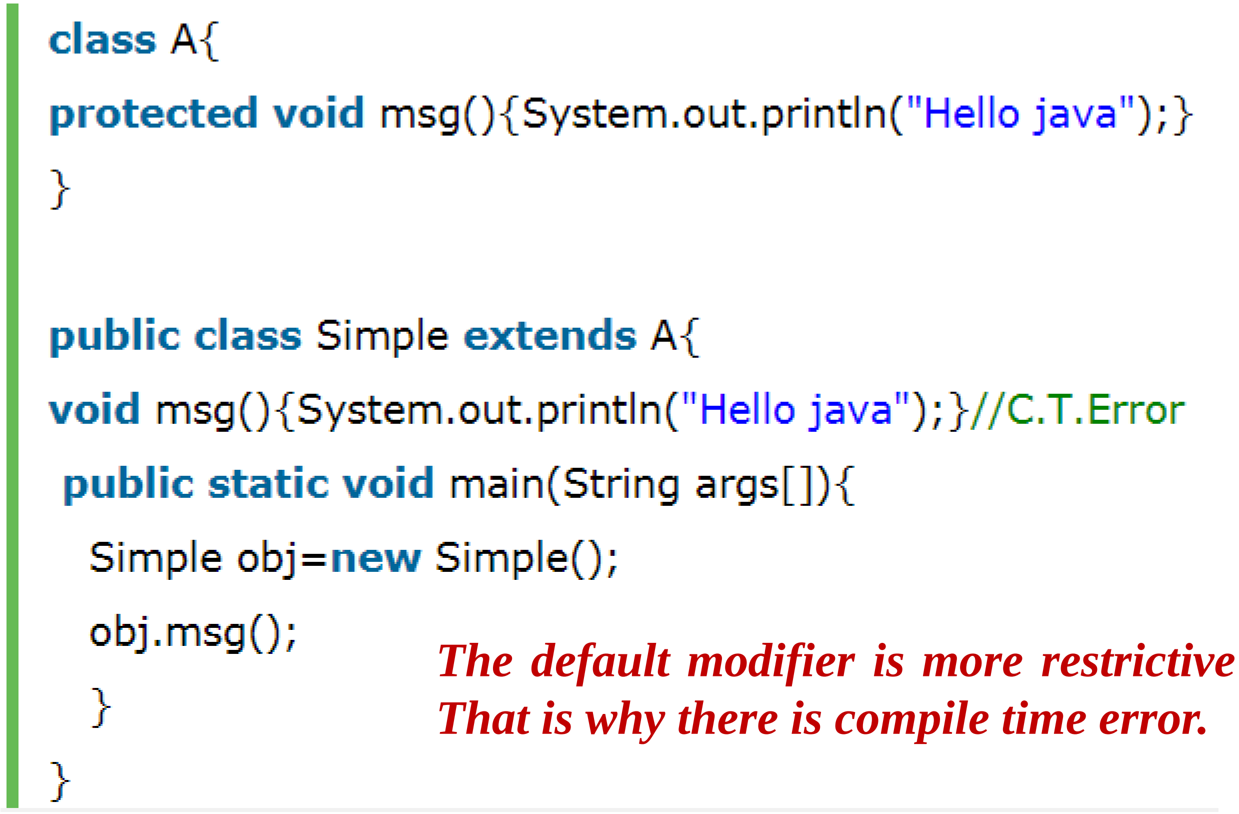


Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Access modifiers & Overriding

- If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.



```
class A{
protected void msg(){System.out.println("Hello java");}
}

public class Simple extends A{
void msg(){System.out.println("Hello java");}//C.T.Error
 public static void main(String args[]){
   Simple obj=new Simple();
   obj.msg();
   }
}
```

The default modifier is more restrictive than protected. That is why there is compile time error.

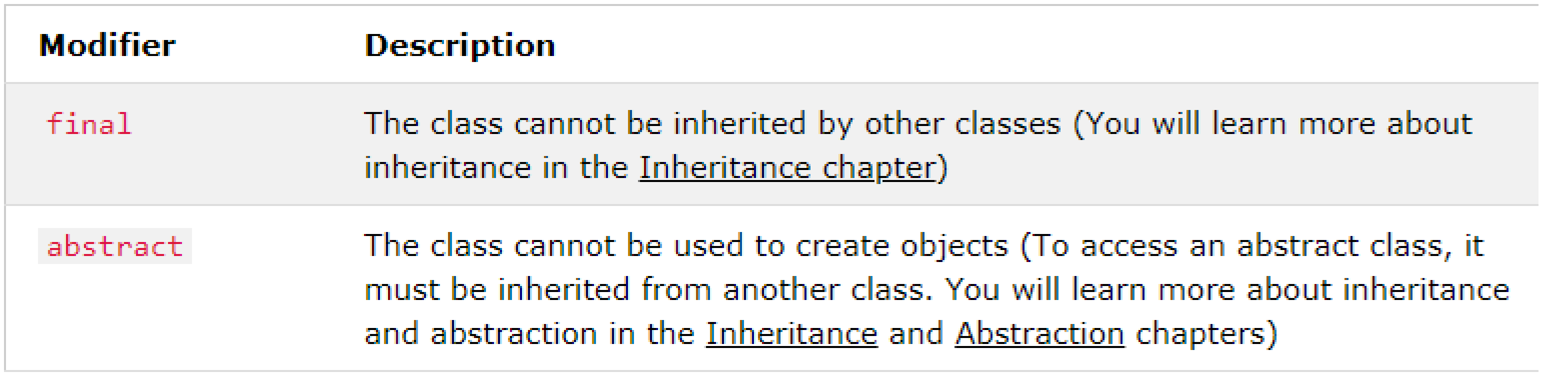
Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Non-access modifiers

Non-Access Modifiers

Do not control access level, but provides other functionality like memory management, restrictions etc.

- For **classes**, you can use either final or abstract

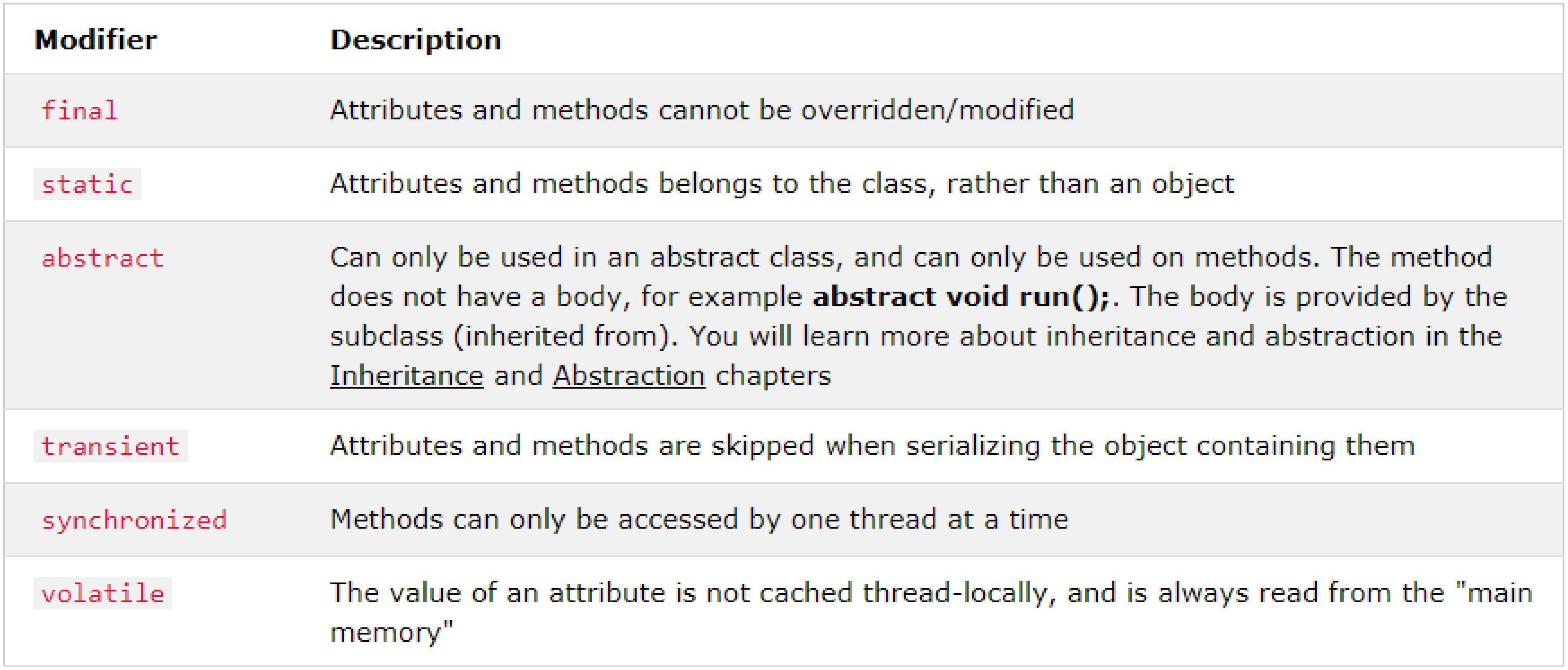


Modifier	Description
`final`	The class cannot be inherited by other classes (You will learn more about inheritance in the Inheritance chapter)
`abstract`	The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters)

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Non-access modifiers

- For **classes**, you can use either final or abstract
- For **attributes and methods**, you can use the one of the following:



Modifier	Description
`final`	Attributes and methods cannot be overridden/modified
`static`	Attributes and methods belongs to the class, rather than an object
`abstract`	Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example **abstract void run();**. The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters
`transient`	Attributes and methods are skipped when serializing the object containing them
`synchronized`	Methods can only be accessed by one thread at a time
`volatile`	The value of an attribute is not cached thread-locally, and is always read from the "main memory"

Here, we will learn access modifiers.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Non-access modifiers

Some common non-access modifiers in java

- **Final**

  If you don't want the ability to override existing attribute values, declare attributes as final:

- **Static**

  A static method means that it can be accessed without creating an object of the class, unlike public:

- **Abstract**

  An abstract method belongs to an abstract class, and it does not have a body. The body is provided by the subclass.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

The final keyword with Class

- If you make any class as final, you cannot extend it.

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

Output: Compile Time Error

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Static: Java static keyword

- The static keyword in Java is used for ***memory management*** mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than an instance of the class.

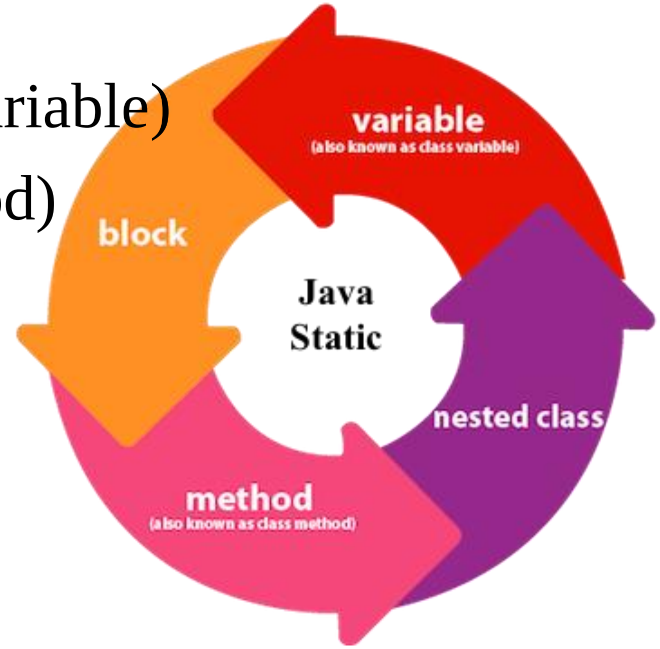
The static can be:

✓ Variable (also known as a instance variable)

✓ Method (also known as a class method)

✓ Block

✓ Nested class



Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

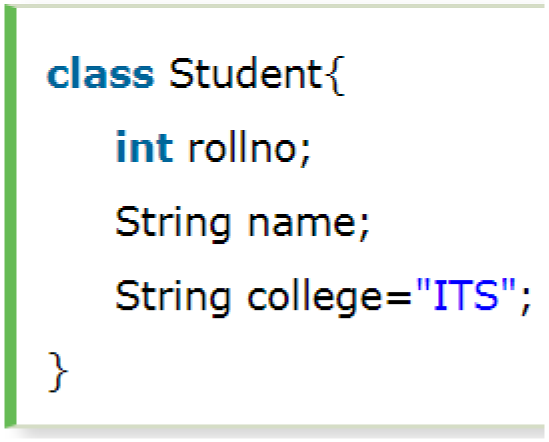
Static: Static Variable

- If you declare any variable as static, it is known as a static variable.
- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

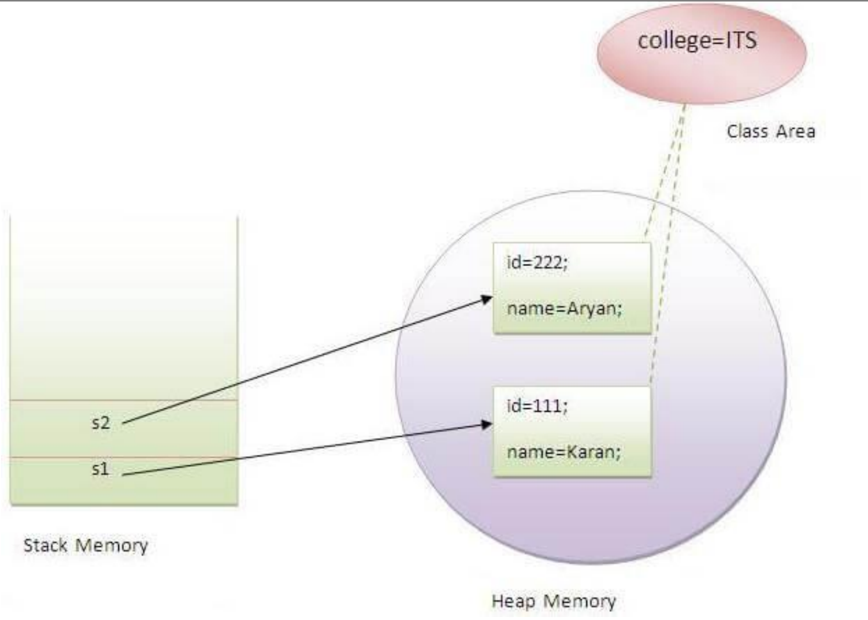
Advantages of static variable:

- ✓ Static Property is share to all object
- ✓ It makes your program **memory efficient** (i.e., it saves memory).

Static: Static Variable



```
class Student{
     int rollno;
     String name;
     String college="ITS";
}
```



- Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique ***rollno*** and ***name***, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Static: Static Methods

- ✓ If you apply static keyword with any method, it is known as static method.
- ✓ A static method belongs to the class rather than the object of a class.
- ✓ A static method can be invoked without the need for creating an instance of a class.
- ✓ A static method can access static data member and can change the value of it.

```
class Calculate{
  static int cube(int x){
  return x*x*x;
  }
```

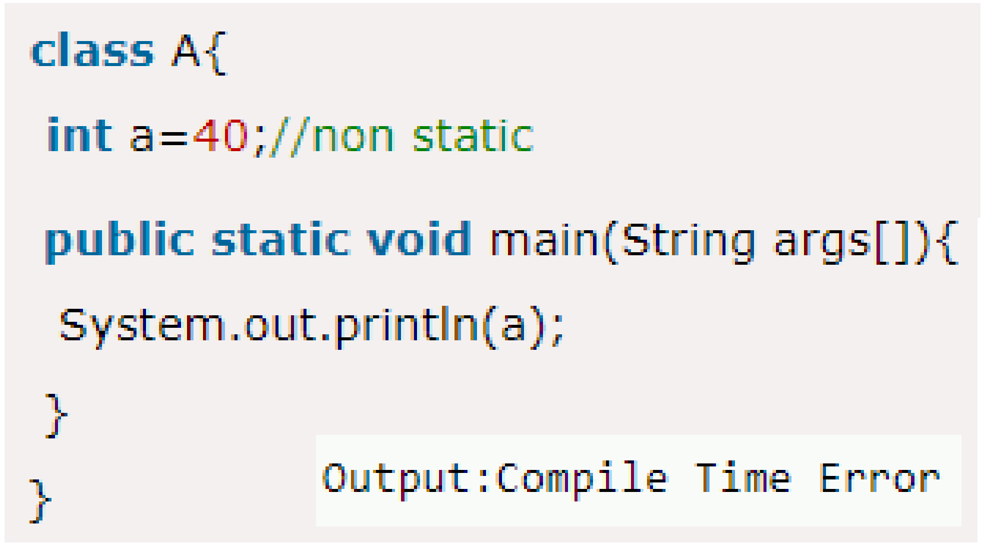
```
public static void main(String args[]){
          int result= cube(5);  or
          int result=Calculate.cube(5);
      System.out.println(result);
  }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Static: Static Methods

Methods declared as **static** have several restrictions:

- They can only directly (*without creating an object*) call other **static** methods.
- They can only directly (*without creating an object*) access **static** data.
- They cannot refer to **this** or **super** in any way.



```
class A{
 int a=40;//non static
 public static void main(String args[]){
  System.out.println(a);
 }
}
```

```
Output:Compile Time Error
```

Static & Non-static variable & method : Access

```
class A{
              static int a=10;
                         int b=20;
              static void show(){ System.out.println(" Static Method ");}
               void hello(){ System.out.println(" Hello Non Static Method “);}

          public static void main(String args[]) {
                  System.out.println(a);
                   show();  // or A.show();

                  A ob=new A();
                 System.out.println(ob.b);
                  ob.hello();
           }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Java Static Properties: variable & method

```
class A{
            static int a=10;
                     int b=20;
            static void show(){ System.out.println(" Static Method ");}
             void hello(){ System.out.println(" Hello Non Static Method “);}
}
```

```
public class Access {
   public static void main(String args[]) {

            System.out.println(a);
            A.show();

            A ob=new A();
             System.out.println(ob.b);
                        ob.hello();
    }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Static: Static Block

- ✓ Is used to initialize the static data member.
- ✓ It is executed before the main method at the time of class loading.

```
class A{
          static{
                   System.out.println("static block is invoked");
        }
        public static void main(String args[]){
                   System.out.println("Hello main");
        }
}
```

Output:

```
static block is invoked
Hello main
```

Static: Static Block

```
// Demonstrate static variables, methods, and blocks.
class StaticBlock {

       static int a = 10;
       static int b=1;

       static void meth(int x) {
              System.out.println("x = " + x);
              System.out.println("a = " + a);
              System.out.println("b = " + b);
       }

      static {
             System.out.println("Static block ");
             b = a * 2;
       }

     public static void main(String args[]) {
                  meth(5);
     }
}
```

- If you declare any block as static then it is called a static block in java.
- *Java Static Block has many interesting properties*
  - Static block is executed at compile time before the execution of main method.
  - You can get output without entering in main method (but there must be a main function)

Output:

```
Static block
x=5
a=10
b=20
```

Java Static Properties: Static Block Example

```
class A{
        static void sheep_talk(){
                System.out.println(" Baaa... ");
                kid_talk()
        }
        static void kid_talk(){
                System.out.println(" Maaa... ");
                sheep_talk();
        }

        static{  kid_talk();  }

       public static void main(String args[]) {
            sheep_talk();
       }
}
```

Output

```
Maaa...
Baaa...
Maaa...
Baaa...
Maaa...
Baaa...
...
...
```

Q:***Can we execute a program without main() method?***

Ans: *No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the main method.*

Varargs: Variable-Length Arguments

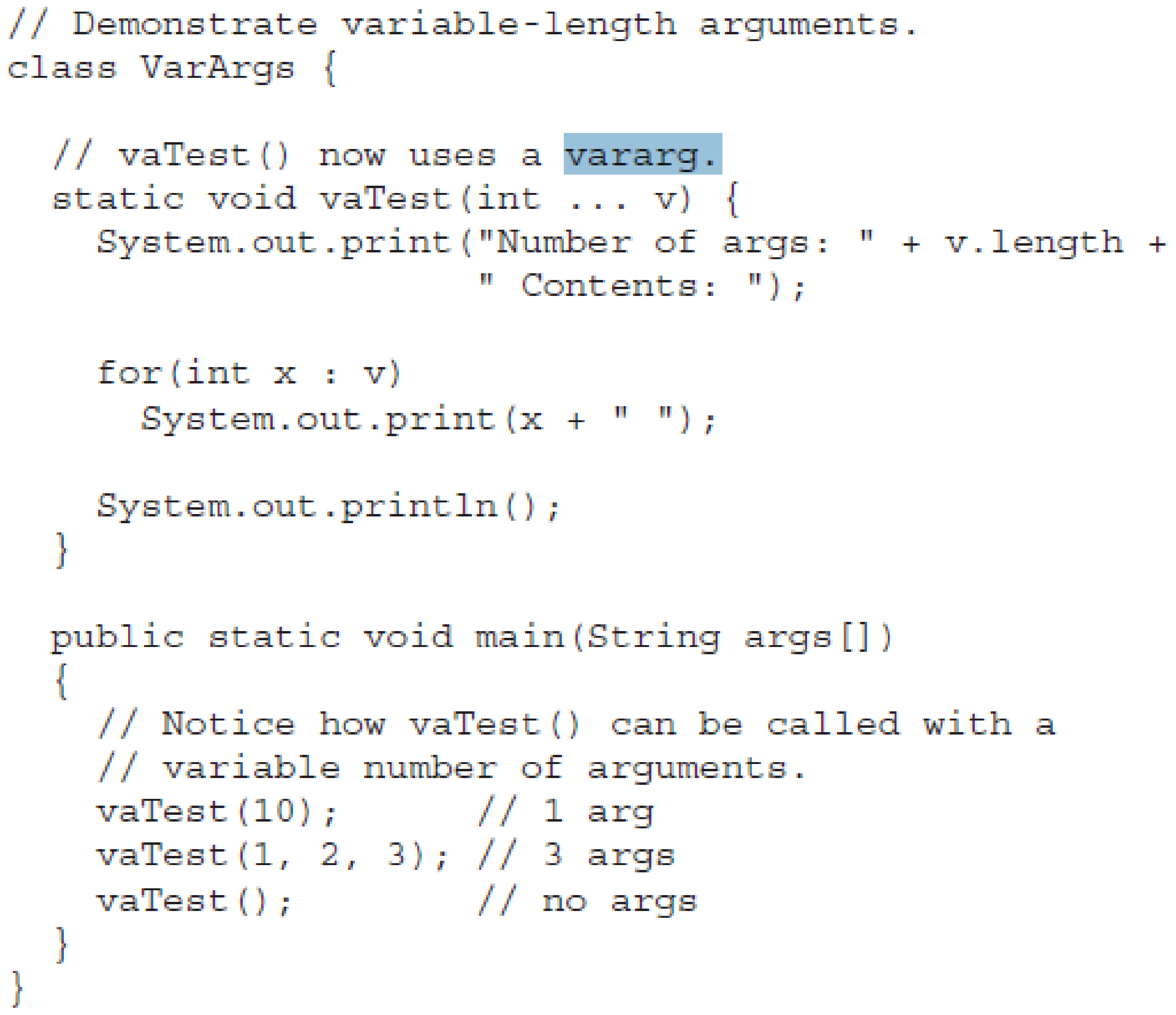
```
// Use an array to pass a variable number of
// arguments to a method. This is the old-style
// approach to variable-length arguments.
class PassArray {
  static void vaTest(int v[]) {
    System.out.print("Number of args: " + v.length +
                     " Contents: ");

    for(int x : v)
      System.out.print(x + " ");
    System.out.println();
  }

  public static void main(String args[])
  {
    // Notice how an array must be created to
    // hold the arguments.
    int n1[] = { 10 };
    int n2[] = { 1, 2, 3 };
    int n3[] = { };

    vaTest(n1); // 1 arg
    vaTest(n2); // 3 args
    vaTest(n3); // no args
  }
}
```

The output from the program is shown here:

```
Number of args: 1 Contents: 10
Number of args: 3 Contents: 1 2 3
Number of args: 0 Contents:
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Varargs: Variable-Length Arguments



```
// Demonstrate variable-length arguments.
class VarArgs {

  // vaTest() now uses a vararg.
  static void vaTest(int ... v) {
    System.out.print("Number of args: " + v.length +
                     " Contents: ");

    for(int x : v)
      System.out.print(x + " ");

    System.out.println();
  }

  public static void main(String args[])
  {
    // Notice how vaTest() can be called with a
    // variable number of arguments.
    vaTest(10);      // 1 arg
    vaTest(1, 2, 3); // 3 args
    vaTest();        // no args
  }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Varargs: Variable-Length Arguments

```
// Use varargs with standard arguments.
class VarArgs2 {

  // Here, msg is a normal parameter and v is a
  // varargs parameter.
  static void vaTest(String msg, int ... v) {
    System.out.print(msg + v.length +
                     " Contents: ");

    for(int x : v)
      System.out.print(x + " ");

    System.out.println();
  }

  public static void main(String args[])
  {
    vaTest("One vararg: ", 10);
    vaTest("Three varargs: ", 1, 2, 3);
    vaTest("No varargs: ");
  }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

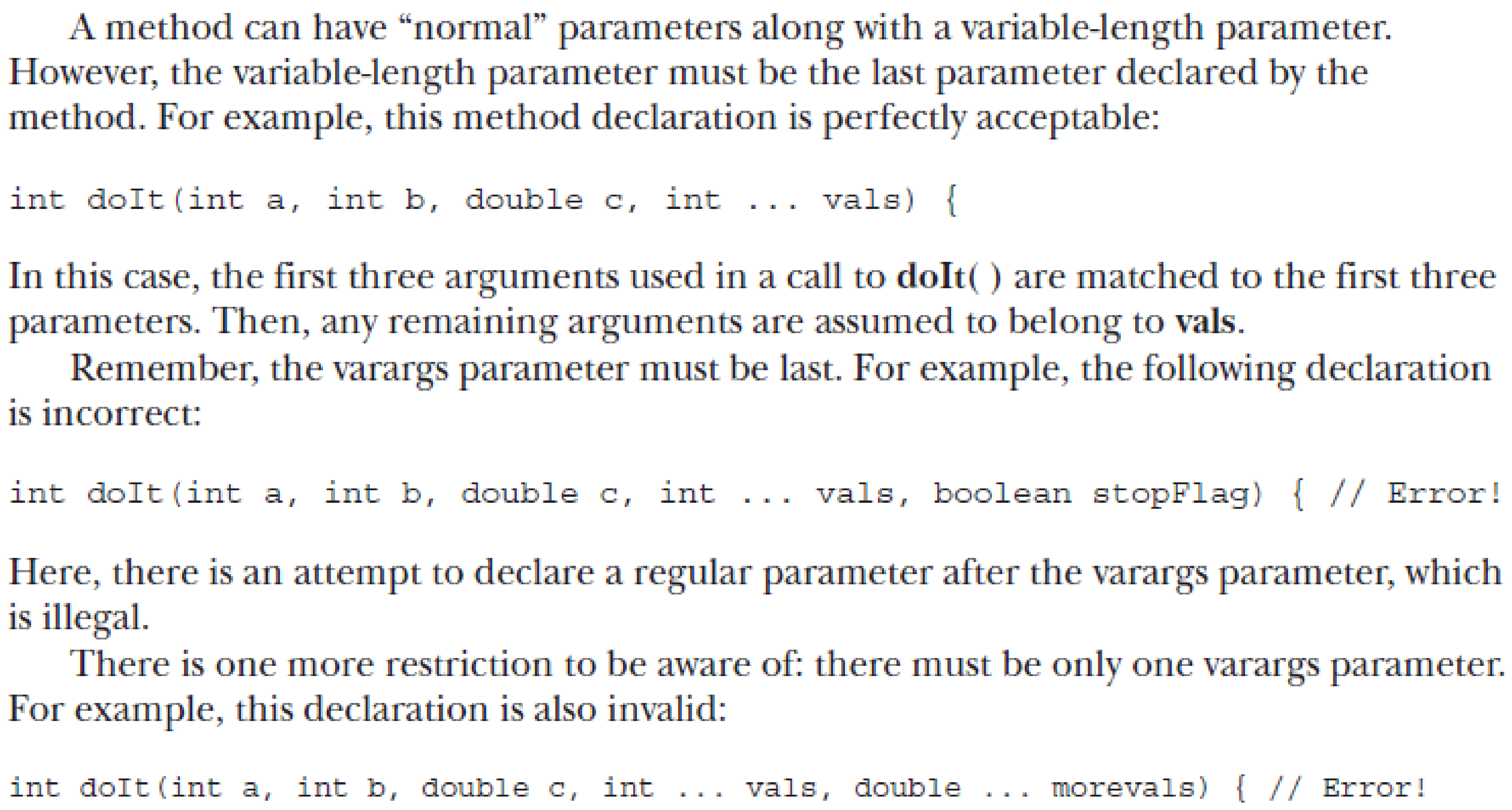
Varargs: Variable-Length Arguments

```java
// Use varargs with standard arguments.
class VarArgs2 {

  // Here, msg is a normal parameter and v is a
  // varargs parameter.
  static void vaTest(String msg, int ... v) {
    System.out.print(msg + v.length +
                     " Contents: ");

    for(int x : v)
      System.out.print(x + " ");

    System.out.println();
  }

  public static void main(String args[])
  {
    vaTest("One vararg: ", 10);
    vaTest("Three varargs: ", 1, 2, 3);
    vaTest("No varargs: ");
  }
}
```

The output from this program is shown here:

```
One vararg: 1 Contents: 10
Three varargs: 3 Contents: 1 2 3
No varargs: 0 Contents:
```

Varargs: Variable-Length Arguments



A method can have "normal" parameters along with a variable-length parameter. However, the variable-length parameter must be the last parameter declared by the method. For example, this method declaration is perfectly acceptable:

```
int doIt(int a, int b, double c, int ... vals) {
```

In this case, the first three arguments used in a call to **doIt( )** are matched to the first three parameters. Then, any remaining arguments are assumed to belong to **vals**.

Remember, the varargs parameter must be last. For example, the following declaration is incorrect:

```
int doIt(int a, int b, double c, int ... vals, boolean stopFlag) { // Error!
```

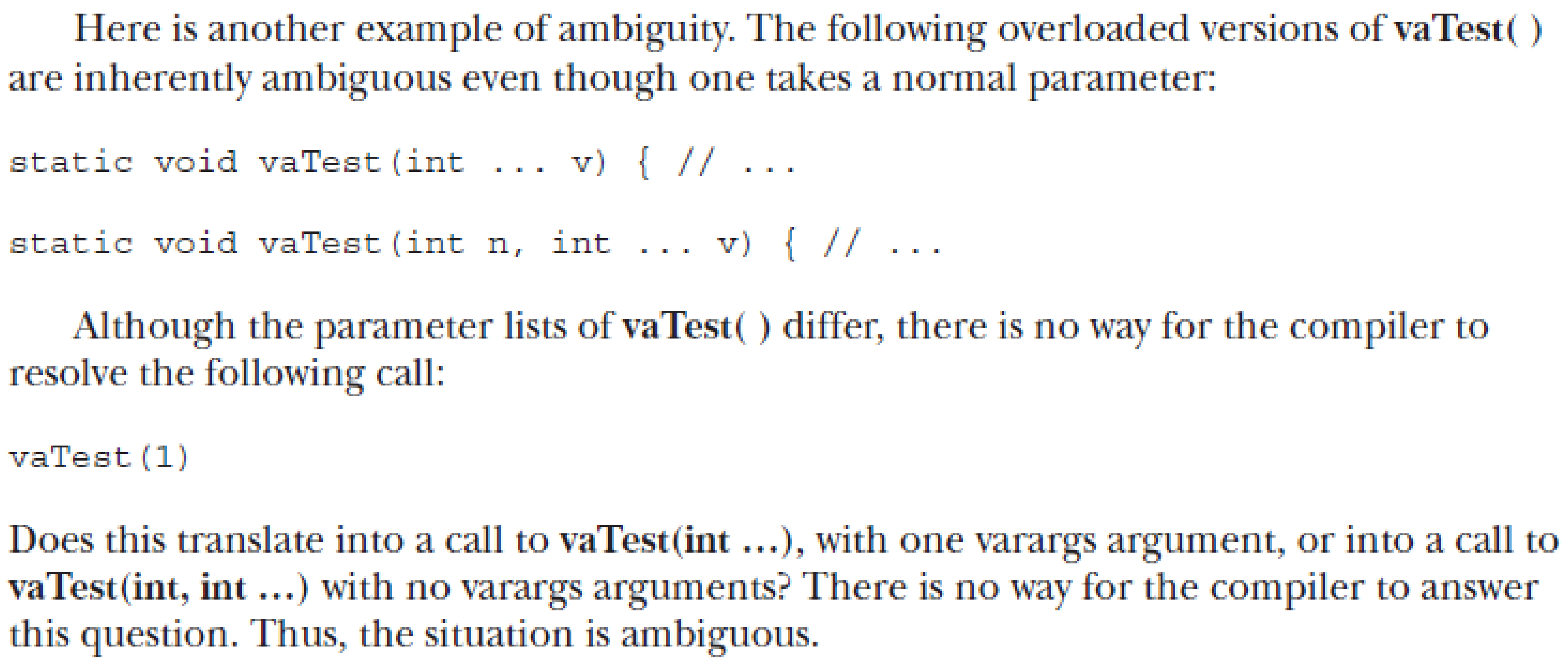
Here, there is an attempt to declare a regular parameter after the varargs parameter, which is illegal.

There is one more restriction to be aware of: there must be only one varargs parameter. For example, this declaration is also invalid:

```
int doIt(int a, int b, double c, int ... vals, double ... morevals) { // Error!
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Varargs and Ambiguity



Here is another example of ambiguity. The following overloaded versions of **vaTest( )** are inherently ambiguous even though one takes a normal parameter:

```
static void vaTest(int ... v) { // ...

static void vaTest(int n, int ... v) { // ...
```

Although the parameter lists of **vaTest( )** differ, there is no way for the compiler to resolve the following call:

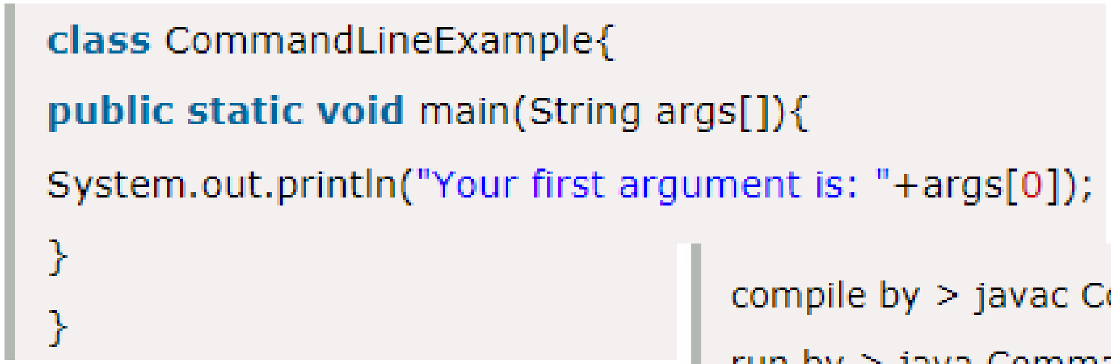
```
vaTest(1)
```

Does this translate into a call to **vaTest(int ...)**, with one varargs argument, or into a call to **vaTest(int, int ...)** with no varargs arguments? There is no way for the compiler to answer this question. Thus, the situation is ambiguous.

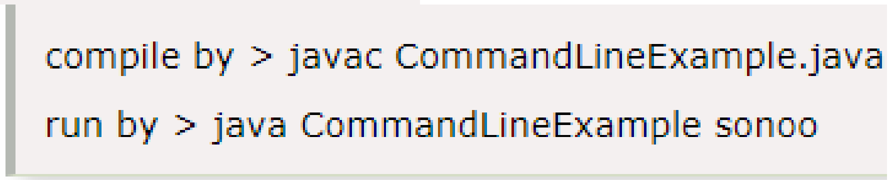
Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Using Command-Line Arguments

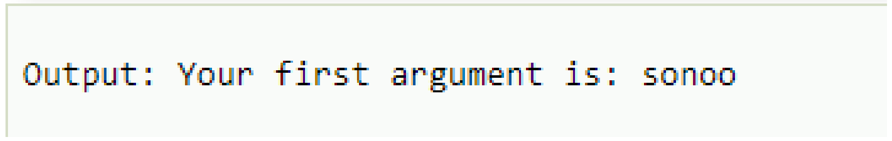
- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behavior of the program for the different values.
- You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.



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

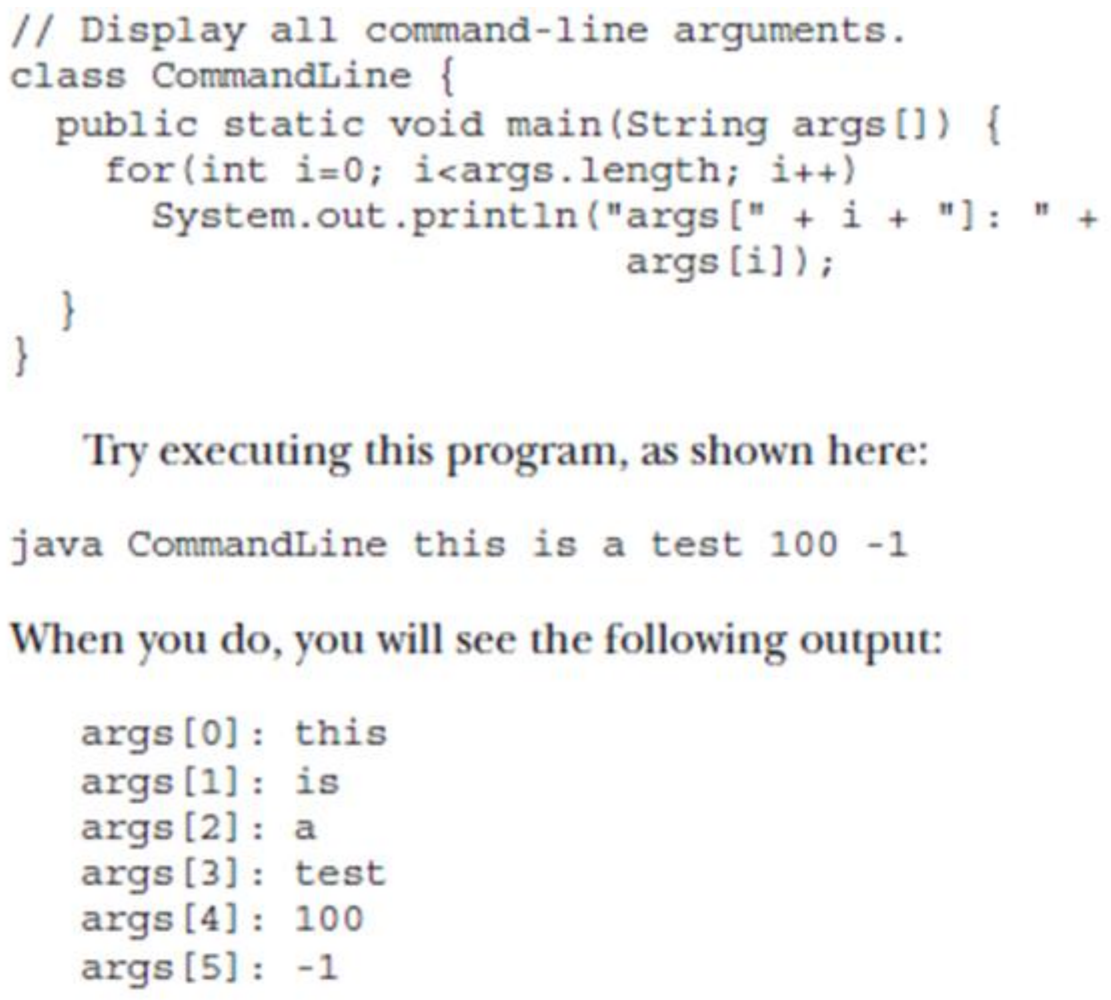


```
compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo
```



```
Output: Your first argument is: sonoo
```

Using Command-Line Arguments



```
// Display all command-line arguments.
class CommandLine {
  public static void main(String args[]) {
    for(int i=0; i<args.length; i++)
      System.out.println("args[" + i + "]: " +
                                  args[i]);
  }
}
```

Try executing this program, as shown here:

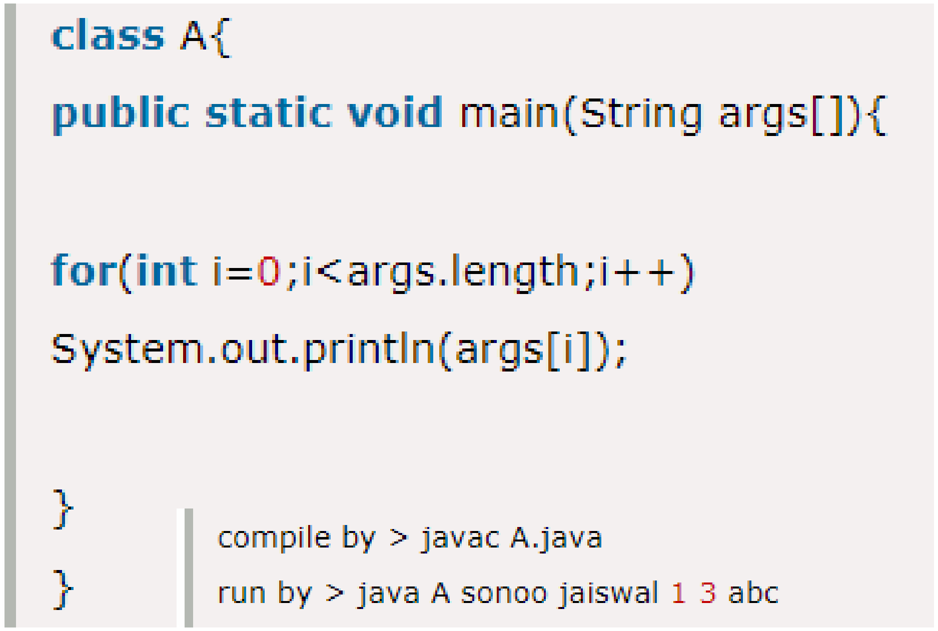
```
java CommandLine this is a test 100 -1
```

When you do, you will see the following output:

```
   args[0]: this
   args[1]: is
   args[2]: a
   args[3]: test
   args[4]: 100
   args[5]: -1
```

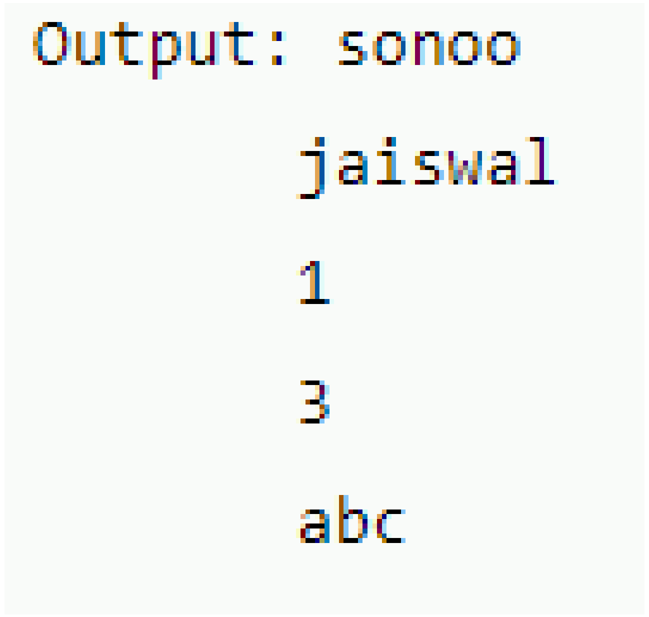
Using Command-Line Arguments

- Example of command-line argument that prints all the values
  - In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.



```
class A{
public static void main(String args[]){

for(int i=0;i<args.length;i++)
System.out.println(args[i]);


}
}
```

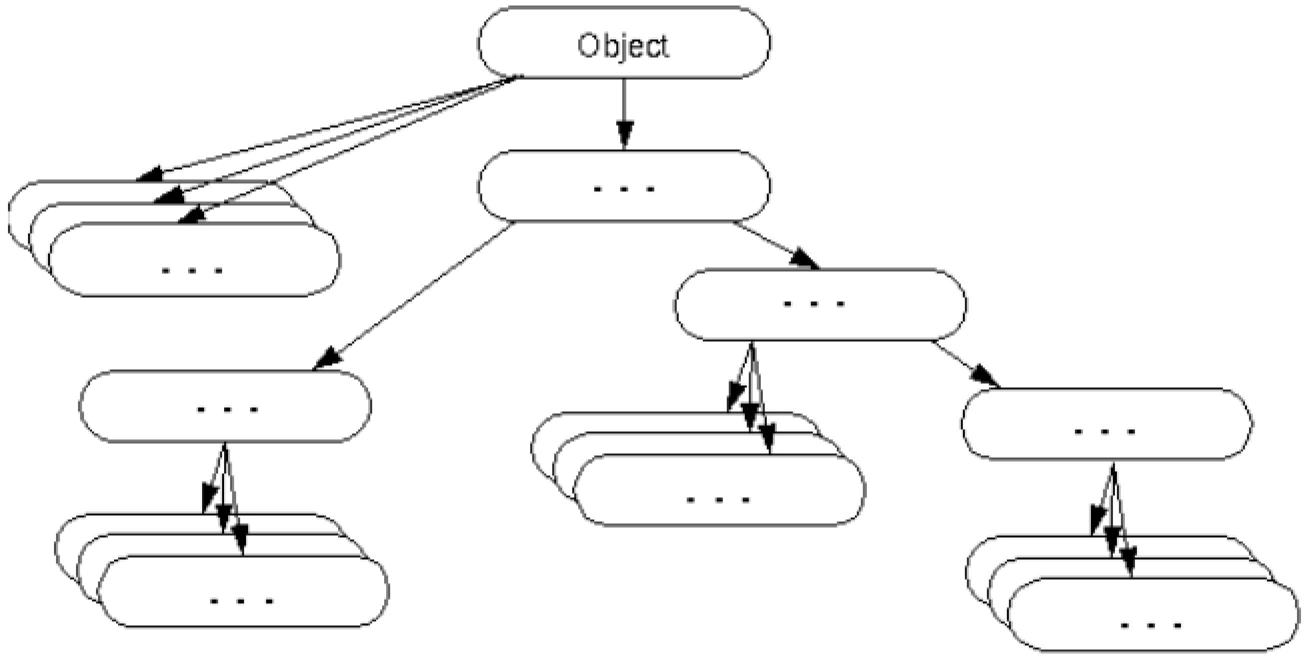
```
compile by > javac A.java
run by > java A sonoo jaiswal 1 3 abc
```



```
Output: sonoo
        jaiswal
        1
        3
        abc
```

The Object Class

- The Object class is the parent class of all the classes in java by default.
- In other words, it is the topmost class of java.



The Object Class (*up-casting*)

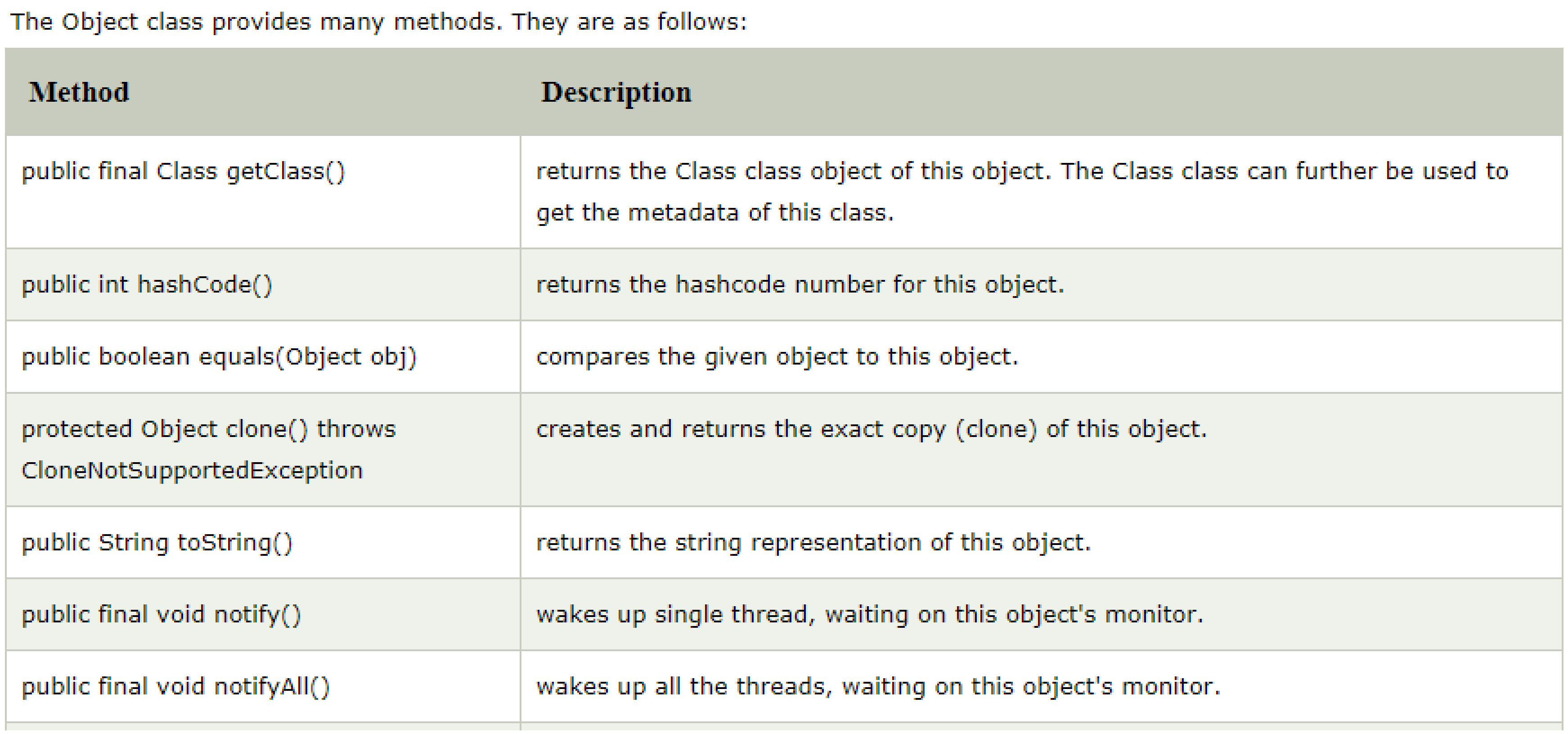
- The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as ***upcasting***.

- Example, there is getObject() method that returns an object but it can be of any type like Emploudent etc, we can use Object class reference to refer that object.

Object obj=getObject(); *//*we don't know what object will be returned from this method

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Some methods of Object Class



The Object class provides many methods. They are as follows:

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.