Module 2

Contents

- Tokens in Java
- Comments in Java
- Variables in Java
- Datatypes in Java
- Operators and expressions in Java
- Decision making and branching
- Classes
- Objects
- Methods

Tokens in Java

- Smallest individual units in a program are known as tokens.
- Java program is a collection of tokens, comments and white spaces

1. Reserved Keywords-must be written in lower case
2. Identifiers-programmer designed tokens.
    a) alphabets,digits,underscore and dollor sign
    b)not begin with digit
    c)upper and lower case letters
    d)any length
3. Literals
4. Operators
5. Separators-parathesis,braces,semicolon,comma,dot

Keywords in Java

```
abstract
assert
boolean
break
byte
case
catch
char
class
const
continue
default
do
double
else
enum
extends
false
final
finally
float
for
goto
if
implements
import
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
strictfp
super
switch
synchronized
this
throw
throws
transient
true
try
void
volatile
while
```

Comments in Java

- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take two forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
symbol, even across line breaks          */
```

Escape sequences in Java

- What if we wanted to print the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```
System.out.println ("I said \"Hello\" to
                    you.");
```

Escape sequences in Java

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

Variables in Java

- Variables are containers for storing data values.
- In Java, there are different **types** of variables, for example:
- **String** - stores text, such as "Hello". String values are surrounded by double quotes
- **int** - stores integers (whole numbers), without decimals, such as 123 or -123
- **float** - stores floating point numbers, with decimals, such as 19.99 or -19.99
- **char** - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- **boolean** - stores values with two states: true or false

- To create a variable, you must specify the type and assign it a value:

  ***type variable* = *value*;**

e.g. String name = "John";
System.out.println(name);

Scope of variables

- Instance variables
- Class variables
- Local variables

Scope of variables

- **Instance** variables:

They are created when objects are instantiated , therefore they are associated with the objects.

- **Class** Variables:

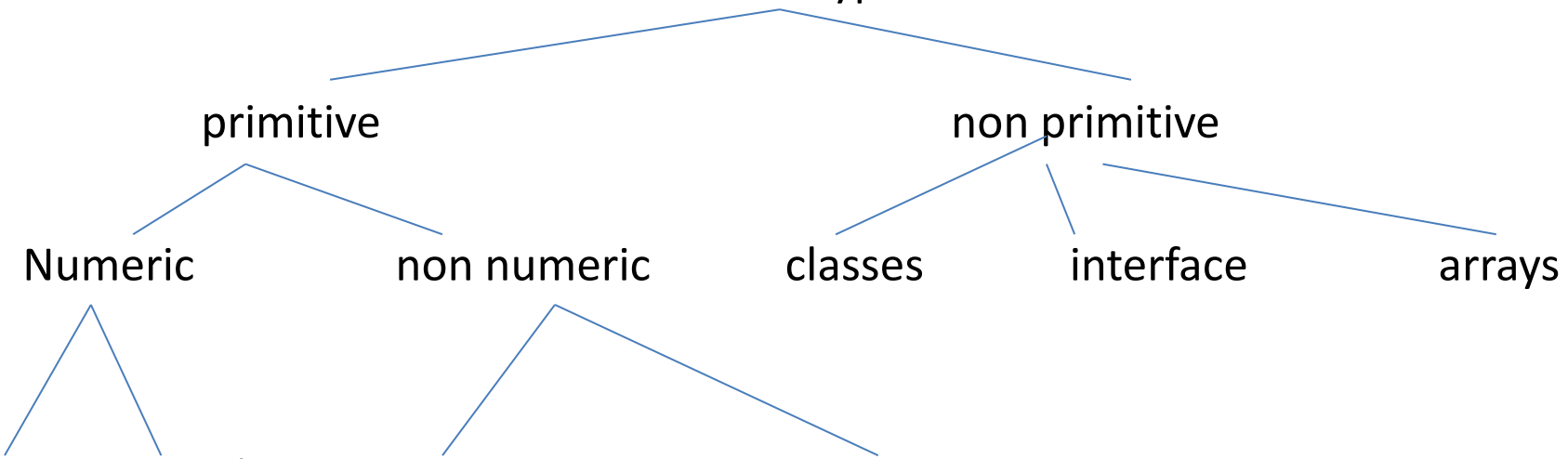
They are global to a class and belong to the entire set of objects that class creates. Only one memory location is created for each class variable.

- **Local** Variables:

They are declared and used inside methods, not available outside the method definition.

Data types

- data types



int float/double char(unicode 16 bit)) boolean

byte (8 bit -128 to 127)

short(16 bit -32768 to 32767)

int(32 bit)

long(64 bit)

JAVA datatypes

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Operators

- Arithmetic(+,-,*,/,%,++,+=,-=,*=,/=,%=,--)
- Bitwise operators(~,&,|,^,>>,<<)
- Logical Operators(&&,||,!)
- Relational operators(==,!=,>,<,>=,<=)
- Ternary Operator

expression1?expression2:expression3

Ex:i=10;

K=i<0?-i:i;

Table 3.11 *Summary of Java Operators*

Operator	*Description*	*Associativity*	*Rank*
. ( ) [ ]	Member selection Function call Array element reference	Left to right	1
- + + - - ! ~ (type)	Unary minus Increment Decrement Logical negation Ones complement Casting	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ -	Addition Subtraction	Left to right	4
<< >> >>>	Left shift Right shift Right shift with zero fill	Left to right	5
< <= > > = instanceof	Less than Less than or equal to Greater than Greater than or equal to Type comparison	Left to right	6
= = ! =	Equality Inequality	Left to right	7
& ^ | && || ?: = op=	Bitwise AND Bitwise XOR Bitwise OR Logical AND Logical OR Conditional operator Assignment operators Shorthand assignment	Left to right Left to right Left to right Left to right Left to right Right to left Right to left	8 9 10 11 12 13 14

It is very important to note carefully, the order of precedence and associativity of operators. Consider the

Accepting input through Scanner class

- Scanner s1=new Scanner(System.in);

- Since Scanner class is defined in util package, so import util package
- Scanner class methods:

String ---objectname.nextLine();

char---objectname.next();

int---objectname.nextInt();

Float---objectname.nextFloat();

double---objectname.nextDouble();

Control structures

Control structures

- The **if** Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

Syntax:

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

Control structures

- The **if…else** Statement:

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax:

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

Control structures

- The **if…else** Statement:

Write a program to accept any number from the user using scanner class and display whether the number is an odd number or an even number.

Control structures

- The switch Statement:

Use the switch statement to select one of many code blocks to be executed.

Syntax:

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Control structures

- The switch Statement:

Write a program to accept any number between 1-12 from the user using scanner class and display which month it is using switch statement.

Control structures

- Loops: **while**

The while loop loops through a block of code as long as a specified condition is true:

Syntax:

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

Control structures

- Loops: **while**

Write a program to calculate sum of first 10 odd numbers.

Control structures

- Loops:The **Do/While** Loop
- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- **Syntax:**

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

Control structures

- Loops: When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop.

Syntax:

for (*statement 1*; *statement 2*; *statement 3*)

```
{
  // code block to be executed
}
```

Control structures

- Loops: for loop

Write a program to find sum of first 10 even numbers, starting from the number which is entered by the user through scanner class.

Control structures

- Loops: For-Each Loop

There is also a "**for-each**" loop, which is used exclusively to loop through elements in an **array**:

Syntax:

for (*type variable* : *arrayname*) {
  // code block to be executed
}

e.g.

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Control structures

break:

- Break is used in switch
- The break statement can also be used to jump out of a **loop**.

e.g.

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}
```

Control structures

- **continue**

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

e.g.

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

Control structures

Write a Program to print 10 Pythagoras Triplets.

Use three Loops(can be For)

Innermost loop will have Break with label.Introduce label to the outermost loop to exit when ten triplets get printed.Pythagoras Triplet

A Triplet where sum of Square of any two Digits is equal the Square of other Digit.

Example

345 => (3*3)+(4*4)=5*5

=> 9 + 16 = 25

=> 25 = 25

Classes, Objects & Methods

Class

Classes, Objects & Methods

- Java is a true object-oriented language and therefore the underlying structure of all Java programs is classes.
- Anything we wish to represent in a Java program must be encapsulated in a class that defines the state and behaviour of the basic program components known as objects.
- Classes create objects and objects use methods to communicate between them.
- Classes provide a convenient method for packing together a group of logically related data items and functions that work on them.
- In Java, data items are called fields and the functions are called methods.
- Calling a specific method in an object is described as sending the object a message.

Classes, Objects & Methods

Defining a class:

- A class is a user-defined data type with a template that serves to define its properties.
- Once the class type has been defined, we can create “variables” of that type using declarations that are similar to the basic type declarations.
- In java, these variables are termed as instances of classes, which are the actual objects.

class definition:

```
class classname [extends superclass]
{
    [field declaration;   ]
    [methods declaration;   ]
 }
```

Note: everything inside the square brackets is optional.

Classes, Objects & Methods

Defining a class:

- A class is a user-defined data type with a template that serves to define its properties.
- Once the class type has been defined, we can create “variables” of that type using declarations that are similar to the basic type declarations.
- In java, these variables are termed as instances of classes, which are the actual objects.

class definition:

```
class classname [extends superclass]
{
    [field declaration;   ]
    [methods declaration;   ]
 }
```

Note: everything inside the square brackets is optional.

Classes, Objects & Methods

Fields declaration:

- Data is encapsulated in a class by placing data fields inside the body of the class definition.
- These variables are called instance variables because they are created whenever an object of the class is instantiated.
- We can declare the instance variables exactly the same way as we declare local variables.

```
class Rectangle
{
  int length;
  int width;
}
```

- The class Rectangle contains two integer type instance variables.
- These variables are only declared and therefore no storage space has been created in the memory.

Methods

Method

Methods

Method declaration:

- A class with only data fields has no life.
- We must therefore add methods that are necessary for manipulating the data contained in the class.
- Methods are declared inside the body of the class but immediately after the declaration of instance variables.
- The general form of a method declaration is

```
type methodname(parameter-list)
{
    method-body;
}
```

Methods

- A **method** is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as **functions**.

Create a Method:

- A method must be declared within a class.
- It is defined with the name of the method, followed by parentheses **().**

```
public class MyClass
{
  static void myMethod()
{
    // code to be executed
  }
}
```

Methods

Call a Method:

- To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**

```
public class MyClass
{
  static void myMethod()
 {
    System.out.println("I just got executed!");
  }

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

Methods

Method Parameters:

- Information can be passed to functions as parameter. Parameters act as variables inside the method.
- Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

```
public class MyClass
 {
  static void myMethod(String fname)
{
    System.out.println(fname);
  }

  public static void main(String[] args)
 {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
```

Methods

Return Values:

- The void keyword, used in the examples above, indicates that the method should not return a value.
- If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method:

```
public class MyClass
 {
  static int myMethod(int x)
 {
    return 5 + x;
  }

  public static void main(String[] args)
{
    System.out.println(myMethod(3));
  }
}
```

Methods

```
public class MyClass
{
  static int myMethod(int x, int y)
   {
    return x + y;
  }

  public static void main(String[] args)
  {
    System.out.println(myMethod(5, 3));
  }
}
```

Recursive Method

- Recursion in java is a **process in which a method calls itself continuously**. A method in java that calls itself is called recursive method. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

```
//Use recursion to add all of the numbers up to 10
public class Main
 {
public static void main(String[] args)
    {
          int result = sum(10);
           System.out.println(result);
     }
static int sum(int k)
  {
   if (k > 0)
      {
         return k + sum(k - 1);
       }
   else
    {
      return 0;
    }
}
 }
```

Objects

- An object in Java is essentially a block of memory that contains space to store all the instance variables.
- Creating an object is also referred to as instantiating an object.
- Objects in Java are created using the **new** operator.
- The **new** operator creates an object of the specified class and returns a reference to that object .
- Here is an example of creating an object of type **Rectangle**.

```
Rectangle rect1;       //declare the object
rect1=new Rectangle();     //instantiate the object
```

- The first statement declares a variable to hold the object reference and the second one actually assigns object reference to the variable.
- The variable **rect1** is now an object of the **Rectangle** class.

```
Rectangle rect1=new Rectangle();
```

Objects

```
//Use recursion to add all of the numbers up to 10(with object)
public class Main
 {
public static void main(String[] args)
    {
          abc s=new abc();
           int result = s.sum(10);
            System.out.println(result);
     }
}
 class abc
{
 int sum(int k)
   {
    if (k > 0)
       {
          return k + sum(k - 1);
        }
    else
     {
       return 0;
     }
   }
 }
```

Objects

```
//Use recursion to add all of the numbers up to 10(without creating an object)
public class Main
 {
public static void main(String[] args)
    {
          int result = abc.sum(10);
           System.out.println(result);
     }
}
class abc
{
static int sum(int k)
  {
   if (k > 0)
      {
         return k + sum(k - 1);
       }
   else
    {
      return 0;
    }
   }
 }
```