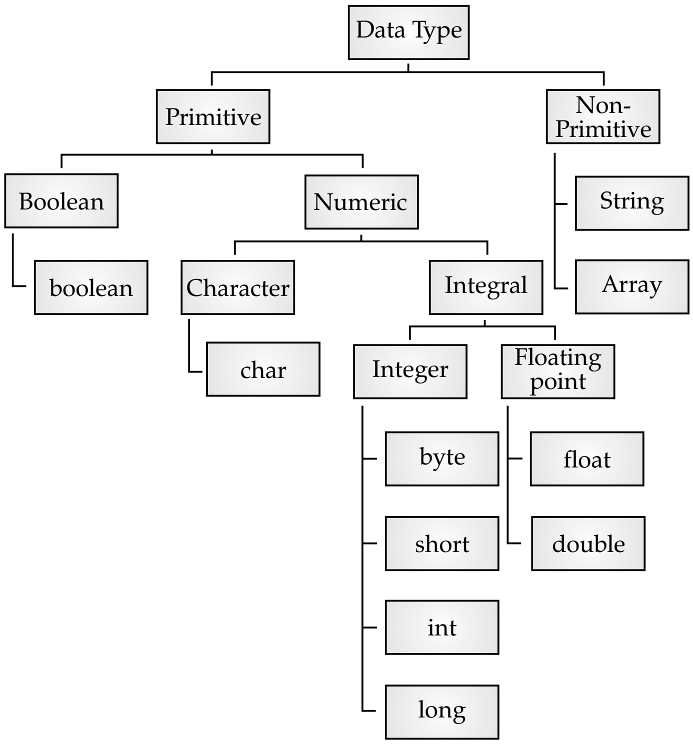
Chapter - 1
REVISION OF CLASS - IX SYLLABUS

Revision Notes

- In Object Oriented programming, the data and its methods are enclosed in a class.
- The basic concepts of OOP include Data Abstraction, Encapsulation, Inheritance, Polymorphism and Modularity.
- Java is an Object-Oriented Language. It is a language that has the Object-Oriented feature, Java supports the following fundamental concepts.
  - **Polymorphism :** Ability of an object to take on many forms.
  - **Inheritance :** A mechanism in which one object acquires all the properties and behaviours of parent class.
  - **Encapsulation :** A process of wrapping code and data together into a single unit i.e., class.
  - **Abstraction :** Process of hiding the implementation details from the user.
- **Object :** Objects have states and behaviours. Example: A dog has states - colour, name, breed as well as behaviours like - wagging the tail, barking, eating etc. An object is an instance of a class.
- **Class :** A class can be defined as a template/blueprint that describes the behaviour/state that the objects of its type support.
- **Instance :** An object is an instance of a class.
- **Method :** A method is a collection of statements that are grouped together to perform the specified task.
- **Local variables :** Variables which are defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables :** Instance variables are variables which are defined within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed inside any method, constructor or blocks of that particular class.
- **Class variables :** Class variables are variables declared within a class, outside any method, with the static keyword.
- **Data Types in Java :**



- **Operators in Java :** The various types of operators and their use are:
  - **Arithmetic Operators:**
    - **Unary operators:** Act on one operand
      - **Unary +** The result is the value of the argument.
      - **Unary -** The result is the negation of the argument.
    - **Binary operators:** Act on two operands

Additional operator +	To add two values.
Subtraction operator -	To subtract two values.
Multiplication operator *	To multiply two values.
Division operator /	To divide two values.
Modulus operator %	To find remainder after dividing two numbers.

  - **Increment/Decrement operators :**

+ +	To increase value of operand by 1
- -	To decrease value of operand by 1

  - **Relational Operators :**

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
= =	Equal to
!=	Not equal to

  - **Logical Operators**
    - **&&** To check truth of both the conditions, acts as AND
    - **||** To check truth of either of the conditions, acts as OR
    - **!** To check falseness of a condition, acts as NOT
    - **^** To check whether the two conditions are different act as XOR
  - **Assignment Operators**
    - = To assign one value to another
- **Library Methods**
  - **abs( ) :** To find absolute value of an argument.
    Syntax : public static datatype abs(argument)
  - **floor( ) :** To find the greatest integer which is less than the given argument.
    Syntax: public static datatype floor(argument)
  - **ceil( ) :** To find the lowest integer which is greater than the given argument.
    Syntax: public static datatype ceil(argument)
  - **pow( ) :** To find value of one argument raised to the power of the other argument.
    Syntax: public static datatype pow(argument1, argument2)
  - **signum( ) :** To find sign (+ or -) of the given argument.
    Syntax: public static datatype signum(argument)
  - **round( ) :** To round off an argument to the nearest decimal.
    Syntax: public static datatype round(argument)
- Iteration statements are statements (or compound statements) to be executed zero or more times, subject to some loop-termination criteria. When these statements are compound statements, they are executed in order, except when either the break statement or the continue statement is encountered.
- Loops are basically meant to do a task multiple times, without actually coding all statements over and over again. *For example, loops can be used for displaying a string many times, for counting numbers and of course for displaying menus.*
- **Loops in Java are mainly of three types :**
  - **'while' loop :** A while statement executes its statements as long as a specified condition evaluates to true.
    Syntax : While (condition)
    statements.
  - **The 'do-while' loop :** It is very similar to the 'while' loop shown above. The only difference being, in a 'while'loop is that the condition is checked before the body of loop, but in a 'do-while' loop, the condition is checked after one execution of the loop.
    Syntax : do statements
    while (condition);
  - **The 'for' loop :** This is probably the most useful and the most used loop in Java. The syntax is slightly more

complicated than that of 'while' or the 'do-while' loop.

Syntax : for(<initial value>;<condition>;<increment/decrement>)
statements.

- **The Break Statement :** Use the break statement to terminate a loop, switch, or in conjunction with a labelled statement.
  - **(i)** When you use break without a label, it terminates the innermost enclosing while, do-while, for, or switch immediately and transfers control to the following statement.
  - **(ii)** When you use break with a label, it terminates the specified labelled statement.

  The syntax of the break statement looks like this :

  break [Label]
- **The Continue Statement :**The continue statement can be used to restart a while, do-while, for or label statement.
  - **(i)** When you use continue without a label, it terminates the current iteration of the innermost enclosing while,do-while, or for statement and continues execution of the loop with the next iteration. In contrast to the break statement, continue does not terminate the execution of the loop entirely. In a while loop, it jumps back to the condition. In a for loop, it jumps to the increment-expression.
  - **(ii)**When you use continue with a label, it applies to the looping statement identified with that label.

  The syntax of the Continue statement looks like the following :

  continue [Label];

Glossary

- **AWT :** Abstract Window Toolkit
- **JVM :** Java Virtual Machine
- **JIT :** Just In Time
- **JDT :** Java Development Toolkit
- **API :** Application Programming Interface

❑❑

Chapter - 2

CLASS AS THE BASIS OF ALL COMPUTATION

Revision Notes

- There are two datatypes available in Java :
  - Primitive Datatypes
  - Reference/Object Datatypes
- **Primitive Datatypes -** There are eight primitive datatypes supported by Java. Primitive datatypes are pre-defined by the language and named by a keyword.
  - **byte**
    - byte datatype is an 8-bit signed two's complement integer.
    - Minimum value is -128 or $(-2)^7$.
    - Maximum value is 127 (inclusive)or $(2^7 - 1)$.
    - Default value is 0.
    - byte datatype is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
    - Example, byte a = 100, byte b = -50.
  - **short**
    - short datatype is a 16-bit signed two's complement integer.
    - Minimum value is -32,768 or $(-2)^{15}$.
    - Maximum value is 32,767 (inclusive) or $(2^{15} - 1)$.
    - Short datatype can also be used to save memory as byte datatype. A short is 2 times smaller than an integer.
    - Default value is 0.
    - *Example*, short s = 10000, short r = -20000.

- **int**
  - int datatype is a 32-bit signed two's complement integer.
  - Minimum value is - 2,147,483,648 or $(-2)^{31}$.
  - Maximum value is 2,147,483,647(inclusive) or $(2^{31}– 1)$.
  - int is generally used as the default datatype for integral values unless there is a concern about memory.
  - The default value is 0.
  - Example, int a = 100000, int *b* = -200000.
- **long**
  - long datatype is a 64-bit signed two's complement integer.
  - Minimum value is -9,223,372,036,854,775,808 or $(-2)^{63}$.
  - Maximum value is 9,223,372,036,854,775,807 (inclusive)$(2^{63}– 1)$.
  - This type is used when a wider range than int is needed.
  - Default value is 0L.
  - Example, long a = 100000L, long b = -200000L.
- **float**
  - float datatype is a single-precision 32-bit IEEE 754 floating point.
  - float is mainly used to save memory in large arrays of floating point numbers.
  - Default value is 0.0f.
  - float datatype is never used for precise values such as currency.
  - *Example*, float f1 = 234.5f.
- **double**
  - double datatype is a double-precision 64-bit IEEE 754 floating point.
  - This datatype is generally used as the default datatype for decimal values, generally the default choice.
  - double datatype should never be used for precise values such as currency.
  - Default value is 0.0d.
  - *Example*, double d1 = 123.4.
- **Boolean**
  - boolean datatype represents one bit of information.
  - There are only two possible values: true and false.
  - This datatype is used for simple flags that track true/false conditions.
  - Default value is false.
  - *Example*, boolean one = true.
- **char**
  - char datatype is a single 16-bit Unicode character.
  - Minimum value is '\u0000' (or 0).
  - Maximum value is '\uffff ' (or 65,535 inclusive).
  - char datatype is used to store any character.
  - *Example*, char letterA = 'A'.

- **Reference Datatypes**
  - Reference variables are created using defined constructors of the classes. They are used to access objects.
  - These variables are declared to be of a specific type that cannot be changed. For *example*, Employee, Puppy, etc.
  - Class objects and various types of array variables come under reference data-type.
  - Default value of any reference variable is null.
  - A reference variable can be used to refer any object of the declared type or any compatible type.
- A variable can be declared as

  <datatype><variable_name>
- An object variable can be declared as

  <class_name><object_name>
- A class is a composite and user-defined datatype.
- In Java, all functionalities are closed in classes.
- When class is used as user-defined data type, it should not include main method in it.
- When a class is to be used as an application, then it should include main method in it.
- Object is the instance of a class.
- Objects are created using the new operator along with the class constructor.

- The real-life objects like cars, dogs, etc. have both state (colour, gears, breed, name, etc.) and behaviour (speed, ignition, halting, barking, running, smelling, etc.).
- Software objects also have state and behaviour. Their state is maintained by variables and behaviour is implemented through methods.
- In an object,
  - There are variables to hold data called member variables or attributes.
  - The values of these member variables define the state of an object.
  - There are member methods / operations / messages that define the behaviour of objects.
- Once a class is declared, the variables of this class type (objects) can be declared and created. So, class can be considered as an object factory.
- Class provides a blueprint for objects. From this blueprint, objects are created using the new operator.
- The keyword *this* refers to a current object which is created and initialized automatically by Java.
- The package contains a set of classes in order to ensure that class names are unique.

Know the Terms

- **Class :** A class is a blueprint that represents a set of similar objects having a common structure and behaviour.
- **Object :** It is an identifiable identity with some characteristics or state and behaviour.

Glossary

- **Byte Code :** Java portable code
- **Applet :** Small Java Program
- **Bean :** Java Software
- **Void :** Null Value
- **Template :** Generic Class

❑❑

Chapter - 3

USER-DEFINED METHODS

Revision Notes

- **Method:** A method is a module which is used simultaneously at different instances in the program is called method or function.
- **Need of methods:**
  - To allow us to handle complex problems.
  - To hide low-level details that otherwise are ambiguous and confuse.
  - To reuse portions of code without retyping it.
- **Syntax of methods:**
  <Access specifier><Return type><method name> (parameter list)
  {• //Body of the method•}
  (i) Access specifier – public or private.
  (ii) Method declared without access specifier is by default treated as public
  (iii) Type- Specifies the data of the value returned from the method
  (iv) Method name – Preferably related to the program
  (v) Parameter list – Variables which receives the value passed from the arguments during method call
- There are two components of a method –
  - Header (also known as method prototype)
  - Body List of parameters is called signature
- Header public intadd() Body return(value). The statement which sends back the value from a method is calledprogram return statement and is used at the end of the program which is a function terminator.
- **Different ways of defining a method:**
  - Receiving value and returning outcome to the caller.
  - Receiving values but not returning outcome to the caller and vice versa.
  - Neither receiving values nor returning outcome to the caller.

- A method is invoked or called by providing the method name followed by the parameters to be sent enclosed in parenthesis.
  For example, a method with prototype as float peri(float l, float b) can be called in the main program as: res = peri(l,b).
- **Formal parameter:**
  - Parameter is a variable used with method signature that receives a value during method call.
  - When the parameter values are received from the caller, then it is called formal parameter.
- **Actual Parameter:** If the values are passed to the method during its call from the caller, then it is actual parameter.
- **Ways to invoke methods:** A method is invoked by two methods call by value or call by method depending upon the ways the arguments are passed to the methods.
- **Call by Value :**
  - Process of passing a copy of actual arguments to the formal parameters.
  - Any change made in the formal will not reflect on actual argument.
- **Call by Reference:**
  - Process of passing the reference of actual arguments to the formal parameters and any change in the formalparameters will be reflected in the actual parameters also.
- A method terminates when either a return statement is encountered or last statement in the method is executed.
- A method may contain several return statements but only one of them gets executed.
- The method of type void does not return a value and thus it cannot be used in expressions.
- **Forms of methods:**
  - Pure method, also called accessor.
  - Impure method, also called mutator.
- Pure method or Accessor takes objects as arguments but does not modify the state of the objects.
- Impure method or mutator takes objects on an argument and changes the state of the objects.
- **Method overloading:**
  - Method with the same name but used for different purposes is called as overloaded methods.
  - It is implemented through polymorphism.
  - Compiler finds the best match of the method arguments and the parameter list during program compilation called static binding.

Know the Terms

- **Module :** A module consists of single block of code.
- **Method prototype :** It is the first line of method definition that tells the program about the type of value returned and number / type of arguments.
- **Method signature :** It refers to the number and types of arguments. It is a part of method prototype.
- **Recursive function :** A function which calls by itself is called as recursive function.
- **Static method :** It belongs to a class which can be called without the object of a class. Static keyword is added before the static method name.
- **Non-static method :** Every method in Java is non-static that can access any static method and static variable without using the object of the class. This method must not have static keyword in the method name.

❑❑

Chapter - 4

CONSTRUCTORS

Revision Notes

- Constructor in Java is a special type of method that is used to initialize the object.
- Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object. That is why, it is known as constructor.

- The two rules to defined a constructor are:-
  - Constructor name must be same as its class name.
  - Constructor must have no explicit return type.
- **Characteristics of Java Constructors**
  - Constructors name must be same as that of its class name.
  - Constructors are called automatically when an object is created.
  - Constructors cannot be private.
  - A constructor can be overloaded.
  - Constructors cannot return a value.
  - Constructors do not have a return type (not even void).
  - Constructors are not members of the class. Thus, they cannot be inherited.
- The main use of Constructors is to initialise the object of that class type with a legal initial value.
- **Types of Java Constructors :** There are following types of constructors:-
  - Default constructor (no-argument constructor)
  - Parameterized constructor
- The parameterized constructors are used to initialize fields of the class with some specific values.
- In Constructor overloading, the number of parameters can be same but it can also have different data types.
- Compiler differentiates constructors on the basis of number of parameters, types of parameters and order of parameters.
- A constructor is a special type of method of a class which is used to initialize a newly created instance of the class.

Know the Terms

- **Default Constructor –** It is the no-argument constructor that is automatically created by compiler in the absence of explicit constructor.
- **Explicit Constructor –** It is the constructor that is created by the programmer.
- **Parameterized Constructor –** This constructor are required to pass parameters on creation of objects.
- **Constructor with Arguments –** This constructor is defined with arguments or parameters.
- **Constructor Overloading –** Constructor overloading is the way of having more than one constructor with different parameters list, in such a way that each constructor performs a different task.

Glossary

- **New:** Java Unary Operator
- **Token:** Internal Value Assigned
- **Separators:** Comma and Terminators
- **Signatures:** Function calling parameters

❑❑

Chapter - 5

LIBRARY CLASSES

Revision Notes

- **Wrapper Classes -** Classes that are used for converting primitive datatypes into objects (also called wrapper objects) are called wrapper classes. For example, char into Character, double into Double, int into Integer etc.
- The primitive data types
  - are defined in the language itself.
  - are not objects
  - do not belong to any class.
- In Java, the data structures accept only objects to store. So, to store datatypes as objects in data structures the wrapper classes are used.

  For example,

  ```
  int x = 15;
  Integer objx = new Integer(x);
  ```

- Wrapper classes also include methods to convert the object into the datatype again.
  For example,
  int y = objx.intValue();
- Primitive datatypes and their wrapper class

Primitive Datatype	Wrapper class
Char	Character
Byte	Byte
short	Short
long	Long
integer	Integer
float	Float
double	Double
boolean	Boolean

- **Autoboxing:** It is the process of automatically converting a primitive type to its corresponding wrapper class object. For example – conversion of int to Integer, double to Double etc.
- **Unboxing:** It is just the reverse process of autoboxing that Automatically converts an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Long to long, Double to double etc.
- There are 8 primitive datatypes defined in Java, they are byte, short, long, float, double, char, boolean.
- Java allows us to create new datatype using combination of the primitive datatypes, such datatypes are called composite or user-defined datatypes.
- Just as variables can be declared of any primitive datatype, similarly we can declare variables of the composite datatypes.
- **Class** is an example of composite datatype and we can declare **objects** as variables of the class.
- **Value-returning methods -** Use a return statement to return the value i.e.,return expression;
  - Each of Java's eight primitive datatypes has a class dedicated to it. These are known as wrapper classes.
  - The most common methods of the Integer wrapper class are summarized in below table. Similar methods forthe other wrapper classes are found in the Java API documentation.

Method	Purpose
int parseInt(String s)	returns a integer value equivalent to string s
long parseLong(String s)	returns a long value equivalent to string s
float parseFloat(String s)	returns a float value equivalent to string s
double parseDouble(String s)	returns a double value equivalent to string s
boolean isDigit(Char ch)	returns a boolean value, that is, true if the given character is a digit and false if the given character is not a digit.
boolean isLetter(Char ch)	returns a boolean value, that is, true if the given character is a letter and false if the given character is not a letter.
boolean isLetterorDigit(Char ch)	returns a boolean value, that is, true if the given character is either a digit or a letter and false if the given character is neither a digit nor a letter.
boolean isLowerCase(Char ch)	returns a boolean value, that is, true if the given character is in lower case and false if the given character is not in lowercase
boolean isUpperCase(Char ch)	returns a boolean value, that is, true if the given character is in upper case and false if the given character is not in upper case.
boolean isWhitespace(Char ch)	returns a boolean value, that is, true if the given character is a blank and false if the given character is not a blank.
char toLowerCase(Char ch)	Converts the specified character to lowercase.
char toUpperCase(Char ch)	Converts the specified character to uppercase.

Know the Terms

➢ **Client of a class:** A program that instantiates objects and calls methods of the class.
➢ **Fields:** A variable that resides within the class, and whose scope is to the entire class.
➢ **Instance variables:** data for each object –
➢ **Class Data:** static data that all objects shared.
➢ **Class Members:** fields and methods, within the class

❑❑

Chapter - 6
ENCAPSULATION

Revision Notes

➢ Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.
➢ Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.
➢ To achieve encapsulation in Java :
  ❍ Declare the variables of a class as private.
  ❍ Provide public setter and getter methods to modify and view the value of variable.
➢ Benefits of Encapsulation :
  ❍ The fields of a class can be made read-only or write-only.
  ❍ A class can have total control over what is stored in its fields.
  ❍ The users of a class do not know how the class stores its data. A class can change the data type of a field and users of the class do not need to change any of their code.
➢ Encapsulation is ineffective in absence of access specifiers.
➢ Access specifiers allow to decide how parts of the classes can be accessed by other classes in other parts of the program. There are four types of access specifiers:
  ❍ Default
  ❍ Private
  ❍ Protected
  ❍ Public
➢ Scope is defined as the program part in which a particular piece of code or a variable can be accessed.
➢ Visibility refers to whether you can use a variable from a given place in a program. It is the accessibility of the variable declared.
➢ Visibility rules

Access Specifier	Accessible by classes in the same package	Accessible by classes in other packages	Accessible by sub-classes in the same package	Accessible by sub-classes in other packages
Default (Package)	✓	×	✓	×
Private	×	×	×	×
Protected	✓	×	✓	✓
Public	✓	✓	✓	✓

➢ Java offers following levels of scope:
  ❍ Data declared within a method or block is called local data or local variable.
  ❍ Data declared at the class level can be used by all methods in that class.
  ❍ Data declared within a method can be used only in that method.
  ❍ Variables declared in inside the blocks are not visible outside of that block.
  ❍ Variables declared outside the blocks are visible inside the blocks.
➢ Instance variables are declared in class outside any method or block. These are declared without the keyword static. These variables can be accessed only by creating objects.

- Static variables are declared using the keyword static within a class outside any method or block. These variables need not be accessed by creating an object of the class but can be accessed with the class_name.variable_name.
- An argument variable is used in Java methods where the number of arguments accepted by a method is not known. This can accept variable number of values and is called varargs.

❑❑

Chapter - 7
ARRAYS

Revision Notes

- Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type.
- An **array** is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Elements of array will be referred as array-name [n], where n is the number of element in the array.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.
- There are two types of array:
  - **Single Dimensional Array** – This type of array contains list of variables of same type that are accessed by a common name. This type of array can have only one full row or one full column.
  - **Multi-Dimensional Array** – These are arrays of arrays. This type of array can have multiple rows and multiple columns.
- **Array Declaration** : Syntax for array declaration

  datatype[] array_name;

  where, datatype can be a primitive datatype like int, cal, Double, byte, etc. or an object.

  array_name is an identifier.
- To allocate memory for array elements,

  array_name= new datatype[size];

  where, array_name can hold <size> elements of defined datatype.
- To declare and allocate memory to an array in one statement, the syntax is:

  datatype[] array_name = new datatype[size];

  For example, Double[] marks= new Double[4];
- An array can be initialized during declaration as Double[] marks = {90, 75, 86.50, 58.25};
- To access each element of an array, for…each loop is used.
- To find the length of an array, length method is used. For example, marks.length.
- Basic Operations on Arrays :
  - **Searching :** It is the process to determine whether the given is present in the array or not. These are of two types –
    - Linear Search
    - Binary Search
  - **Sorting :** The process of arranging data in ascending or descending order. There are two ways to sort the data –
    - Selection Sort
    - Bubble Sort
- **Array as composite type :** Arrays contain other values and do not have a standard size. So, they require more memory than primitive types. Java handles arrays by reference.

Know The Term

Array Element : An individual element in the array.

❑❑

Chapter - 8
STRING HANDLING

Revision Notes

Method	Description
String trim()	removes beginning and ending spaces of this string.
String toLowerCase()	returns a string in lowercase.
String toUpperCase()	returns a string in uppercase.
int length()	returns string length
char charAt(int n)	returns char value for the particular index n
intindexOf(int char)	returns the specified char value index.
String concat(String str)	concatenates the specified string.
boolean equals(String str)	checks the equality of string with the another given string.
booleanequalsIgnoreCase(String str)	checks the equality of string with the another given string irrespective of the case.
booleanstartsWith(String str)	compares the two strings based on the Unicode value of each character in the strings.
intcompareToIgnoreCase(String string)	Same as CompareTo method however it ignores the case during comparison.
String replace(char oldChar, char newChar)	returns the new updated string after changing all the occurrences of oldChar with the newChar.
String substring(intbeginIndex)	returns the substring of the string. The substring starts with the character at the specified index.
String substring(intbeginIndex, intendIndex)	Returns the substring. The substring starts with character at beginIndex and ends with the character at endIndex.
intcompareTo(String string)	tests whether the string is having specified str, if yes then it returns true else false.
booleanendsWith(String suffix)	Checks whether the string ends with the specified suffix.

❑❑