DATA TYPE	
int	Eg:
double	
boolean	
char	
String	
comments	Single comment
	Multiple Comment
Loops	

for loop	for(starting index;ending index;increment)
	{
	//code

while loop	while(condition)
	{
	//code
	}

do while	do
	{
	//code
	}while(condition)

switch	switch (expression){
	  case value1:
	  statement1;
	  break;
	
	  case valueN:
	  statementN;
	  break;
	  default:
	  default statement;
	}

Access Modifiers	private
	default
	protected
	public

if condition	if(condition) {
	statement; //executes when condition is true
	}

if else condition	if(condition) {
	statement; //executes when condition is true
	}
	else{
	statement; //executes when condition is false
	}

Method declaration	AccessModifier returnType methodName{
	//code
	}

static keyword	static methodName(){

Method parameters	AccessModifier returnType methodName(parameter 1, parameter 2...){
	//code
	}

Arrays	DataType[] arrayName=new DataType[arraySi

try catch	try {
	  // Block of code to try
	}
	catch(Exception e) {
	  // Block of code to handle errors
	}

break	jump-statement; break;
continue	jump-statement; continue;

inheritance	class Subclass-name extends Superclass-name
	{
	  //methods and fields
	}

Polymorphism	

Method Overloading	AccessModifier returnType methodName(parameter 1, parameter 2...){
	//code
	}
	AccessModifier returnType methodName(parameter 3, parameter 4...){
	//code
	}

Method Overriding	Class className{
	Method(){
	}
	Class className1 extends className{
	Method(){
	//diiferent implementation
	}

interface	interface <interface_name>{
	
	// declare constant fields
	// declare methods that abstract
	// by default.
	}

Abstract	
ArrayList	**ArrayList<String> arraylistName = new**

LinkedList	**LinkedList<String> linklistName = new Linked**
HashMap	**HashMap<String, String> hashmapName**
HashSet	**HashSet<String> setName = new HashSet<Stri**

iterator	
Date Function	
String Functions	
charAt()	Returns the character at the specified index (position)
compareTo()	Compares two strings lexicographically
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences
concat()	Appends a string to the end of another string
contains()	Checks whether a string contains a sequence of characters

endsWith()	**Checks whether a string ends with the specified character(s)**
equals()	**Compares two strings. Returns true if the strings are equal, and false if not**
equalsIgnoreCase()	**Compares two strings, ignoring case considerations**
indexOf()	**Returns the position of the first found occurrence of specified characters in a string**
isEmpty()	**Checks whether a string is empty or not**
lastIndexOf()	**Returns the position of the last found occurrence of specified characters in a string**
length()	**Returns the length of a specified string**
replace()	**Searches a string for a specified value, and returns a new string where the specified values are replaced**
replaceAll()	**Replaces each substring of this string that matches the given regular expression with the given replacement**
split()	**Splits a string into an array of substrings**
startsWith()	**Checks whether a string starts with specified characters**

subSequence()	**Returns a new character sequence that is a subsequence of this sequence**
substring()	**Returns a new string which is the substring of a specified string**
toCharArray()	**Converts this string to a new character array**
toLowerCase()	**Converts a string to lower case letters**
toString()	**Returns the value of a String object**
toUpperCase()	**Converts a string to upper case letters**
trim()	**Removes whitespace from both ends of a string**

int a=10;		
double b=23.45		
boolean c=true;		
char d='A';		
String v="Welcome"		
//This is sample comment		
/* This is multiple comment */		

for(int a=0;a<10;a++
{
//code		

int a-=1;
while(a<10)
{
a++;
}		

int a-=1;
do
{
a++;
}while(a<10)		

char grade='A'; **switch(grade){** **case 'A':** **//code** **default:** **//code** **}**		
A private modifier's access level is restricted to members of the class. **It isn't accessible outside of the class.**		
A default modifier's access level is limited to the package. **It's not possible to get to it from outside the package.** **If you don't indicate an access level, the default will be used.**		
A protected modifier's access level is both within and outside the package via a child class.		
A public modifier's access level is universal. **It can be accessed from within and outside the class, and from within and outside the package.**		

public void method1{ } public int method2{ } public string method3{ } public boolean method4{ }	public static void method1{ } public static int method2{ } public static string method3{ } public static boolean method4{ }	
	methods,	

public void method1(dataType Variable, dataType variable){
}
public int method2(dataType Variable, dataType variable){*
}
public string method3(dataType Variable, dataType variable){{ }
public boolean method4(dataType Variable, dataType variable){{ }
	public static void method1(dataType Variable, dataType variable){ }
	public static int method2(dataType Variable, dataType variable){* }
	public static string method3(dataType Variable, dataType variable){{ }
	public static boolean method4(dataType Variable, dataType variable){{ }	

ze];		

for(int i=1;i<=10;i++){
if(i==5){
//breaking the loop
break;
}
	break keyword will stop the execution	

for(int i=1;i<=10;i++){
if(i==5){
//using continue statement
continue;//it will skip the rest statement
}
	continue keyword will skip the condition	

class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}		

class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
	Method names can be same,
	method names can be same but
	different arguments/parameters	

class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
	Method overlading occurs in different class and
	has a relationship(inheritance)	

interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}
 	
public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}		

abstract class Bike{
  Bike(){System.out.println("bike is created");}
  abstract void run();
  void changeGear(){System.out.println("gear changed");}
}
//Creating a Child class which inherits Abstract class
class Honda extends Bike{
void run(){System.out.println("running safely..");}
}
//Creating a Test class which calls abstract and non-abstract methods
class TestAbstraction2{
public static void main(String args[]){
Bike obj = new Honda();
obj.run();
obj.changeGear();
}
}		

ArrayList<String> cars = new ArrayList<String>();
  cars.add("Volvo");
  cars.add("BMW");
  cars.add("Ford");
  cars.add("Mazda");
  System.out.println(cars)		

LinkedList<String> cars = new LinkedList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
System.out.println(cars);		

HashMap<String, String> capitalCities = new
HashMap<String, String>();
 	
// Add keys and values (Country, City)
capitalCities.put("England", "London");
capitalCities.put("Germany", "Berlin");
capitalCities.put("Norway", "Oslo");
capitalCities.put("USA", "Washington DC");
System.out.println(capitalCities);		

HashSet<String> cars = new HashSet<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("BMW");
cars.add("Mazda");
System.out.println(cars);		

HashSet<String> cars = new HashSet<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
 	
// Get the iterator
Iterator<String> it = cars.iterator();
 	
// Print the first item
System.out.println(it.next());
	ArrayList<String> cars = new ArrayList<String>();
	cars.add("Volvo");
	cars.add("BMW");
	cars.add("Ford");
	cars.add("Mazda");
	
	// Get the iterator
	Iterator<String> it = cars.iterator();
	
	// Print the first item
	System.out.println(it.next());
		LinkedList<String> cars = new LinkedList<String>();
		cars.add("Volvo");
		cars.add("BMW");
		cars.add("Ford");
		cars.add("Mazda");
		
		// Get the iterator
		Iterator<String> it = cars.iterator();
		
		// Print the first item

LocalDate myObj = LocalDate.now(); // Create a date object
System.out.println(myObj); // Display the current date		

char		
int		
int		
String		
boolean		

boolean		
boolean		
boolean		
int		
boolean		
int		
int		
String		
String		
String[]		
boolean		

CharSequence		
String		
char[]		
String		
String		
String		
String		