Objetos	
Instancias	En los lenguajes de programación orientada a objetos un objeto es una instancia de una clase. Esto es, un miembro de una clase que tiene atributos en lugar de variables.
Atributos	Los atributos son las caracterísiticas individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades. Los atributos se guardan en variables denominadas de instancia, y cada objeto particular puede tener valores distintos para estas variables. Las variables de instancia también denominados miembros dato, son declaradas en la clase pero sus valores son fijados y cambiados en el objeto. Además de las variables de instancia hay variables de clase, las cuales se aplican a la clase y a todas sus instancias. Por ejemplo, el número de ruedas de un automóvil es el mismo cuatro, para todos los automóviles.
Constructores	Un constructor inicializa un objeto al crearlo. Tiene el mismo nombre que su clase y se parece sintácticamente a un método, pero no tienen tipo de return. Normalmente se usan para dar valores iniciales a las variables de instancia definidas por la clase, o para llevar a cabo otros procedimientos de inicio para crear un objeto completamente formado. todas las clases tienen constructores, los definas o no, porque java ofrece un constructor por defecto que inicializa todas las variables de miembro a 0. `class ClassName { ClassName() { } }`

-No argument Constructors	using these constructors the instance variables of a method will be initialized with fixed values for all objects.
	`Public class MyClass { Int num; MyClass() { num = 100; } }`
	You would call constructor to initialize objects as follows
	`public class ConsDemo { public static void main(String args[]) { MyClass t1 = new MyClass(); MyClass t2 = new MyClass(); System.out.println(t1.num + " " + t2.num); } }`
	
	Se pueden modificar los atributos de la clase creando una instancia, un nuevo objeto, y accediendo a ellos main{ Humidade h = new Humidade(); h.num = 1000; }
	Humidade int num = 10; Humidade(){num = 100;} El roden es 1000, 100, 10

Objetos (cont)

-Parameterized Constructors	Most often, you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method.
	`// A simple constructor. class MyClass { int x; // Following is the constructor MyClass(int i ) { x = i; } }`
	You would call constructor to initialize objects as follows
	`public class ConsDemo { public static void main(String args[]) { MyClass t1 = new MyClass( 10 ); MyClass t2 = new MyClass( 20 ); System.out.println(t1.x + " " + t2.x); } }`

"this" keyword	Keyword THIS is a reference variable in Java that refers to the current object.
Variables de instancia	Una variable de instancia es un campo declarado dentro de la declaración de una clase sin usar la palabra reservada static. Si una clase T tiene un campo a que es una variable de instancia, entonces una nueva variable de instancia a es creada e inicializada a un valor por defecto como parde de cada nuevo objeto creado de la clase T o de cualquier clase que sea subclase de T La bariable de instancia deja de existir cuando el objeto del cual es campo deja de ser referenciado, después que cualquier finalización necesaria del objeto ha sido completada
variables de clase	Una variable de clase es un campo declarado usando la palabra reservada static dentro de una declaración de clase o con o sin la palabra reservada static dentro de una declaración de una interfaz. Una variable de clase es creada cuando su clase o interfaz es preparada y es inicializada a un valor por defecto. La variable de clase deja de existir cuando su clase o interfaz es descargada.

Métodos

A method in Java is a block of statements that has a name and can be executed by calling (also called invoking) it from some other place in your program. Along with fields, methods are one of the two elements that are considered members of a class. (Constructors and initializers are not considered class members.).

Every program must have at least one method for the program to accomplish any work. And every program must have a method named main, which is the method first invoked when the program is run.

```
visibility [static] return-type method-name (parameter-list) { statements... }
```

Métodos (cont)

visibility: The visibility of a method determines whether the method is available to other classes. The options are	-**public**: Allows any other class to access the method -**private**: Hides the method from other classes -**protected**: Lets subclasses use the method but hides the method from other classes
static	: This optional keyword declares that the method is a static method, which means that you can call it without first creating an instance of the class in which it's defined. The main method must always be static, and any other methods in the class that contains the main method usually should be static as well. A non-static method belongs to an object of the class and you have to create an instance of the class to access it.

return-type: After the word static comes the return type, which indicates whether the method returns a value when it is called — and if so, what type the value is. If the method doesn't return a value, specify void. String[], Integer, int...

parameter list:En el momento de la creación se deb especificar el tipo de la variable y la variable.Al llamar al método se especifica el nombre.	public static int METODO(String[] nomes){

Pasar por argumento:

-

-

Reference types

Reference types hold references to objects and provide a means to access those objects stored somewhere in memory.

All reference types are a subclass of type java.lang.Object.

Annotation	
Array	Provides a fixed-size data structure that stores data elements of the same type.
Class	Designed to provide inheritance, polymorphism, and encapsulation. Usually models something in the real world and consists of a set of values that holds data and a set of methods that operates on the data.
Enumeration	
Interface	Provides a public API and is "implemented" by Java classes.

https://www.oreilly.com/library/view/java-8-pocket/9781491901083/ch04.html

Lists	
Arrays	*var-name = new type [size];*
	int intArray[]; intArray = new int[20]; // alocar memoria al array
	int[] intArray = new int[20]; // combining both statements in one
Arrays bidimensionales	int[][] multi = new int[5][10];
	int[][] multi = new int[5][]; multi[0] = new int[10]; multi[1] = new int[10]; multi[2] = new int[10]; multi[3] = new int[10]; multi[4] = new int[10];
	int tiempo [][] = new int[7][2];
	tiempo[0][1] = 2;
	int[][] multi = new int[][]{ { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };
	for(int i = 0; i < t.length; i++){ System.out.println("Día "+i);//el día de la semana for(int j = 0; j < 2; j++){ System.out.println("a las "+j+"h: "+t[i][j]);//j - la hora } }
Arraylist class	implements List interface and it is based on an Array data structure
	ArrayList inherits AbstractList class and implements List interface.
	ArrayList is initialized by a size, however the size can increase if collection grows or shrunk if objects are removed from the collection.
	ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases

Lists (cont)

	ArrayList(): This constructor is used to build an empty array list
	ArrayList(Collection c): This constructor is used to build an array list initialized with the elements from collection c
	ArrayList(int capacity): This constructor is used to build an array list with initial capacity being specified // Creating generic integer ArrayList ArrayList<Integer> arrli = new ArrayList<Integer> ();
	arrli.add(i);
	// Remove element at index 3 arrli.remove(3);
	// Printing elements one by one for (int i=0; i<arrli.size(); i++) System.out.print(arrli.get(i)+" ");

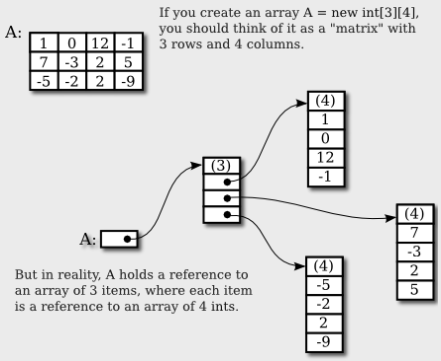
Los ArrayList funcionan con objetos y no con tipos primitivos, (Integer en vez de int, Character en vez de char...))	
Construye ArrayList con valor inicial	ArrayList(int initialCapacity)
Inserta el elemento en el index especificado, no se peude hacer si la posición no fue creada	add(int index, Object element)
Appends the specified element to the end of this ArrayList.	add(Object o)
	addAll(Collection c) Appends all of the elements in the specified Collection to the end of this this ArrayList, in the order that they are returned by the specified Collection's Iterator.
Borra todos los elementos	clear()
hace una copia superficial	clone()
Devuelve true si contiene el elemento	contains(Object elem)
Devuelve el elemento en la posición especificada	cars.get(0);
Modificar elemento	cars.set(0, "Opel");
Busca la ocurrencia del primer elemento	indexOf(Object elem)
	isEmpty()
devuelve el índice de la última ocurrencia	lastIndexOf(Object elem)
Quita el elemento del índice especificado	cars.remove(0)
Returns the number of components in this ArrayList.	size()
	toArray() ,] toArray(Object[] a)
Trims the capacity of this ArrayList to be the ArrayList's current size.	trimToSize()

Lists (cont)

Pasando por argumento	public void AnalyseArray(ArrayList<Integer> array) { // Do something } ... ArrayList<Integer> A = new ArrayList-<Integer>(); AnalyseArray(A);
Bidimensionales Crear:	ArrayList[][] table = new ArrayList[10][10]; table[0][0] = new ArrayList(); // add another ArrayList object to [0,0] table[0][0].add(); // add object to that ArrayList
Sacar tamaño del ArrayList	int size = listOfBanks.size();
Sacar tamaño de ArrayList bidimensional	list.get(0).get(0)
Añadir elementos	arrlist.add(15); // adding element 25 at third position arrlist.add(2,25);
Pasar arraylist bidimensional por método	ArrayList<ArrayList<Integer>> disponibilidade
Agregar valor en array bidimensional	ArrayList<ArrayList<Integer>> d = new ArrayList<ArrayList<Integer>>(); for(int i = 0; i < d.size(); i++){ for(int j = 0; j < d.get(i).size(); j++){ d.get(i).add(0); }

https://www.geeksforgeeks.org/arraylist-in-java/
https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/util/ArrayList.html
https://www.programiz.com/java-programming/multidimensional-array

Arrays multidimensionales



Classes

Variables

Globales	Java doesn't technically support global variables. As a pure object-oriented language, everything needs to be part of a class. The static modifier tells Java that the variable can be used across all instances of the class.

If

If	if (a > b) { max = a; } else { max = b; }
Operador ternario	max = (a > b) ? a : b;
Booleanos	if(!contiene){}

Switch case

int c; switch(c){ case 1: case 2: case 3: resultado = x; case 4: resultado = y; default: resultado = z;}

Bucles

for simplificado	for (String str : list) { System.out.println(str); } Object[] humidades = new Object[10]; for(Object h : humidades){
for	for (int i = 0; i < list.size(); i++) { System.out.println(list.get(i)); }

Packages

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc. A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and namespace management.

Clase Scanner

Para string	Scanner sc = new Scanner(System.in);
	String a = sc.nextLine();
Para char	c = sc.next().charAt(0);
Después de un scan.nextInt()	sc.nextLine();

Tipos primitivos

En Java hayq ue especificar lso tipos primitivos antes que usarlos.

String	String.length(); String usa " "
char	char usa ' '
Para forzar tipos (explicit casting)	int randomNumber = (int)(Math.random());

Interfaces

Un interfaz es una lista de acciones que puede llevar a cabo un determinado objeto. En una clase además de aparecer los métodos aparecía el código para dichos métodos, en cambio en un interfaz sólo existe el prototipo de una función, no su código.
En java un interfaz define la lista de métodos, pero para que una clase posea un interfaz hay que indicar explícitamente que lo implementa mediante la claúsula implements.

Donde modif.visibilidad puede ser public o bien sin especificar, es decir visibilidad pública (desde cualquier clase se puede emplear el interfaz) o de paquete (sólo se puede emplear desde clases del mismo paquete).
nombreInterfaz por convenio sigue las mismas reglas de nomenclatura que las clases, y en muchos casos acaba en able (que podíamos traducir como: ser capaz de).
-La claúsula opcional extends se emplea para conseguir que un interfaz hereda las funciones de otro/s interfaces, simplemente listaInterfaces es una lista separaada por coma de interfaces de los que se desea heredar.
-

```
[modif.visibilidad] interface nombreInt-
erfaz [extends listaInterfaces] { prototipo
método1; .. prototipo método1; }
```

Interfaces (cont)

En muchas ocasiones un interfaz es empleado para definir un comportamiento, que posteriormente será implementado por diversas clases, que podrían no tener nada que ver entre ellas, pero que todas se comportarán igual de cara al interfaz. Es decir, todas tendrán las funciones indicadas por el interfaz.

Cuando varios objetos de distintas clases pueden responder al mismo mensaje (función), aún realizando cosas distintas se denomina **polimorfismo**.

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface can contain any number of methods.

An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.

The byte code of an interface appears in a .class file.

Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

You cannot instantiate an interface.

An interface does not contain any constructors.

All of the methods in an interface are abstract.

An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.

An interface is not extended by a class; it is implemented by a class.

An interface can extend multiple interfaces.

contrato entre as clases concretas que a implementen, xa que a clase que o faga atópase obrigada a definir os métodos abstractos que a compoñen.

Podemos dicir que as interfaces simulan a herdanza múltiple xa que unha clase pode implementar calquera numero de interfaces, ademais as interfaces poden herdar un ou mais números de interfaces mediante a palabra extends

Todos os métodos dunha interfaz son implicitamente public abstract, non é necesario especificalo na declaración do mesmo.

http://www.mundojava.net/interfaces.html?Pg=java_inicial_4_5.html

Java Cheat Sheet

by Sara (lasago) via cheatography.com/70899/cs/18394/

Ficheros

Crear ficheros

Escribir fichero	File f = new File(rutayarchivo);

Escribir objetos a fichero

Primero hay que implementar la clase Serializable desde la clase del objeto	`public class Student implements Serializable {}`
Luego, abrir o crear un nuevo fichero con FileOutputStream	`FileOutputStream fileOut = new FileOutputStream(filepath);`
Crear un ObjectOutputStream dándole el anterior FileOutputStream como argumento al constructor.	`ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);`
Usar el método ObjectOutputStream.writeObject para escribir el objeto que quieras al fichero.	`objectOut.writeObject(serObj);`
	for(int i=0;i<arrayper.size();i++) { oos.writeObject(arrayper.get(i)); }
Cerrar el flujo	objectOut.close();

```
package com.javacodegeeks.java.core; import java.io.FileOutputStream; import java.io.ObjectO-
utputStream; public class ObjectIOExample { private static final String filepath="C:\\Users\\n-
ikos7\\Desktop\\obj"; public static void main(String args[]) { ObjectIOExample objectIO = new
ObjectIOExample(); Student student = new Student("John","Frost",22); objectIO.WriteObjectToFi-
le(student); } public void WriteObjectToFile(Object serObj) { try { FileOutputStream fileOut =
new FileOutputStream(filepath); ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
objectOut.writeObject(serObj); objectOut.close(); System.out.println("The Object was succes-
fully written to a file"); } catch (Exception ex) { ex.printStackTrace(); } } }
```

Recuperar los atributos del objeto	`public static Object crearObjeto( float altitud, int valor ){ Humidade h = new Humidade( altitud, valor ); System.out.println(" "+h+""+h.altitude+" "+h.valor); return h; }`

Recuerda guardar los métodos de fichero en un try catch

Escribir tetxo a ficheros

Creando fichero al abrir el flujo	`FileOutputStream foos = new FileOutputStream(new File("texto.txt"));`

Ficheros (cont)	
Usando fichero existe-nte/cerar uno nuevo	FileOutputStream foos = new FileOutputStream("texto2.txt");
Escribiendo la cadena en el fichero	foos.write(cadena.getBytes("utf-8"), 0, cadena.length());

	`Primer parámetro: cadena a escribir,` **`getBytes():`** `This method encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.`
	`Segundo parámetro: lugar desde donde se dejó de escribir`
	`Tercer parámetro: número de carácteres a escribir`

Escribir float, etc a fichero (en binario)	http://www.java2s.com/Tutorial/Java/0180__File/WritefloattoafileusingDataOutputStream.htm
Leer objeto de fichero	
Abrir flujo	FileInputStream fis = new FileInputStream("meteo.dat");
Abrir input de objeto	ObjectInputStream ois = new ObjectInputStream(fis);
	Object obj = ois.readObject(); Persoa l = (Persoa) ois.readObject();
	ObjectInputStream ois = new ObjectInputStream(fis); for (int i = 0; i <persoas.size(); i++){ Persoa per = (Persoa) ois.readObject(); System.out.println(""+per.apelido+" y "+per.cp); persoasLeidas.add(per); }
Leer varios objetos de un fichero	catch (FileNotFoundException ex) { Logger.getLogger(E3E1E2.class.getName()).log(Level.SEVERE, null, ex); }`

	try {
	boolean engadir = true;
	FileInputStream fis = new FileInputStream("meteo.dat"); ObjectInputStream ois = new ObjectInputStream(fis);
	while(engadir){
	Object obj = ois.readObject();
	if(obj != null){objetosLeidos.add(obj);
	else{engadir = false; }
	for(int i = 0; i < objetosLeidos.size(); i++){
	System.out.println("Objeto leído "+objetosLeidos.get(i));}}

Hacer casting de un tipo objeto con el objeto que queremos	Humidade h = (Humidade) humidades.get(i);

Ficheros (cont)

	public static ArrayList LER() throws FileNotFoundException, IOException, ClassNotFoundException{ FileInputStream fis = new FileInputStream("casas.dat"); ObjectInputStream ois = new ObjectInputStream(fis); boolean saidaDatos = true; ArrayList<Casa> casasleidas = new ArrayList<>(); try{ while(saidaDatos){ Casa casa = (Casa) ois.readObject(); System.out.println("Casa leida: "+casa+" ID "+casa.id+" NH "+casa.hab+" M "+casa.m2); casasleidas.add(casa); } }catch(EOFException ex){ System.out.println("no hay más que leer"); } return casasleidas; }

Ficheros de acceso aleatorio

Random access files permit nonsequential, or random, access to a file's contents. To access a file randomly, you open the file, seek a particular location, and read from or write to that file.

Crea un flujo de fichero de acceso aleatorio desde el que leer y, opcionalmente, al que escribir el archivo especificado por el argumento de Fichero/el archivo con el nombre especificado	RandomAccessFile(File file, String mode)

The java.io.RandomAccessFile.seek(long pos) method sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.	raf.seek(0);
	System.out.println("Elemento a visualizar"); int x = sc.nextInt(); RandomAccessFile raf = new RandomAccessFile("indices.ind", "r"); raf.seek(x); int posicion = raf.readInt(); RandomAccessFile rafO = new RandomAccessFile("datos.dat", "r"); rafO.seek(posicion); System.out.println(""+rafO.readUTF());

https://examples.javacodegeeks.com/core-java/io/fileoutputstream/how-to-write-an-object-to-file-in-java/

Superclase

Subclases

Determinar la clase de un objeto

Herencia

This allows you to establish a hierarchy for your classes. A class that inherits from some other class (referred to as a superclass) is called a subclass. While a subclass inherits methods and behaviors from a superclass, it can also declare new fields and methods (as well as override superclass methods).

The subclass inherits state and behavior in the form of variables and methods from its superclass. The subclass can use just the items inherited from its superclass as is, or the subclass can modify or override it.

Subclases	A subclass is defined with the extends keyword. For example, the syntax ClassB extends ClassA establishes ClassB as a subclass of of ClassA. Java only supports single inheritance, meaning a subclass cannot extend more than one superclass.

Constructores de subclases	Because a constructor initializes an instance of a class, they are never inherited; however, the subclass must call a superclass constructor as it is an extension of a superclass object. This can be done in either of the two ways shown below.
	class MySuperclass{ // superclass instance variable: String myString; // superclass default (empty) constructor: MySuperclass(){} // superclass parameterized constructor: MySuperclass(String myString){ // initialize instance variable this.myString = myString; } }
	1) The subclass makes an explicit call to the superclass' parameterized constructor (i.e.: it calls super(...);): class MySubclass extends MySuperclass{ // subclass constructor: MySubclass(String myString){ // explicit call to superclass constructor: super(myString); } }
	2) The subclass makes an implicit call to the superclass' default constructor (i.e.: a behind-the-scenes call to super(); happens automatically): class MySubclass extends MySuperclass{ MySubclass(String myString){ // behind-the-scenes implicit call to superclass' default constructor happens // subclass can now initialize superclass instance variable: this.myString = myString; } }
	In the second example above, observe that we are initializing a field (myString) that isn't even declared in that class; the reason why this works is because it's inherited from MySuperclass and therefore can be accessed with the this keyword.

Herencia (cont)

Overriding	If a class inherits a method from its superclass, then there is a chance to override the method provided that it is not marked final. class Animal { public void move() { System.out.println("Animals can move"); } } class Dog extends Animal { public void move() { System.out.println("Dogs can walk and run"); } } public class TestDog { public static void main(String args[]) { Animal a = new Animal(); // Animal reference and object Animal b = new Dog(); // Animal reference but Dog object a.move(); // runs the method in Animal class b.move(); // runs the method in Dog class } } This program will throw a compile time error since b's reference type Animal doesn't have a method by the name of bark.https://www.tutorialspoint.com/java/java_overriding.htm
	As a subclass, your class inherits member variables and methods from its superclass. Your class can choose to hide variables or override methods inherited from its superclass.
	las subclases no heredan métodos privados
	se pueden usar métodos protected

	public class Oral extends Examen{
	int temporesp;
	@Override
	public void ALTA(){
	super.ALTA();
	Scanner sc = new Scanner(System.in);
	System.out.println("tempo de resposta");
	this.temporesp = sc.nextInt();
	}

Abstracción

Defien modelo de objeto
Las clases abstractas pueden o no tener métodos abstractos, métodos sin cuerpo (public void get();)
Las clases abstractas pueden tener también constructores class TimesTwo extends Product { public TimesTwo() { super(2); } }
pero si tiene al menos un método abstracto deben ser clases abstractas
No se pueden instanciar
para suar una clase abstracta tienes que heredarla desde otra clase, y ofrecer implementaciones para sus métodos abstractos
Los métodos abstractos no tienen cuerpo y demandan que sus subclases incluyan ese método
Todas las clases que hereden de la clase deben o Override el método abstracto o declararse ellas como abstractas

Abstracción (cont)

public abstract class Animal {
String name;
abstract String sound(); //all classes that implement Animal must have a sound method }

public class Cat extends Animal {
public Cat() {
this.name = "Garfield";
}
@Override
public String sound(){ //implemented sound method from the abstract class & method
return "Meow!";
}
}

A diferencia de las interfaces, no tienes por qué implementar todos los métodos

If the subclass has a method with the same name as the parent's method that the subclass extends, the subclass' method overwrites the parent. If you want to use the parent class's method instead, you use the super keyword, like this: super.startRobot();

Métodos de cadena

X2 = sc.nextLine(); X2 = X2.toUpperCase();

Polimorfismo

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

class Animal { public void animalSound() { System.out.println("The animal makes a sound"); } } class Pig extends Animal { public void animal-Sound() { System.out.println("The pig says: wee wee"); } } class Dog extends Animal { public void animalSound() { System.out.println("The dog says: bow wow"); } }

Abstracción

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods

// Abstract class abstract class Animal { // Abstract method (does not have a body) public abstract void animalSound(); // Regular method public void sleep() { System.out.println("Zzz"); } } // Subclass (inherit from Animal) class Pig extends Animal { public void animalSound() { // The body of animalSound() is provided here System.out.println("The pig says: wee wee"); } }

Interface

	An interface is a completely "abstract class" that is used to group related methods with empty bodies:
	// interface interface Animal { public void animalSound(); // interface method (does not have a body) public void run(); // interface method (does not have a body) }
	To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:
multiple interfaces	// DemoClass "implements" FirstInterface and SecondInterface class DemoClass implements FirstInterface, SecondInterface {

Java Enums

An enum is a special "class" that represents a group of constants (unchangeable variables, like final variables).

enum Level { LOW, MEDIUM, HIGH }

public class MyClass { enum Level {

You can also have an enum inside a class:

By **Sara** (lasago)
cheatography.com/lasago/

Not published yet.
Last updated 5th June, 2019.