Centro de Investigación en Computación del IPN

Java - curso básico

Dra. Erandi Castillo Montiel

Arreglos

Un arreglo o Array, es un objeto en el que se pueden agrupar un número de objetos o datos primitivos , del mismo tipo.
Cada uno de los objetos dentro el array, tiene un identificador numérico único asignado dependiendo de sus posición en el arreglo.

Sintaxis:

tipo [] variable_arreglo;
tipo variable_arreglo[];

Ejemplo:

int [] arrI;
String [] arrS;
float arrf[];

Dimensionando un arreglo

Consiste en crear un arreglo con el tamaño que tendrá

variable_array= new tipo[tamaño];

*Los indices de un arreglo se encuentran entre 0 y tamaño-1

arrI.length;

```
arrI = new int [10];
arrS = new String[5];
arrf= new float[10];
```

```
int numeros [] = new int [10];


for (int i =0; i<numeros.length; i++){
    numeros[i]=i*2;
    System.out.println(numeros[i]);
}
```

Array como argumento

Sintaxis:

Como argumento:

```
tipo_retorno nombreMetodo (tipo []arreglo){
//sentencias
}
```

Retorno de un arreglo:

```
tipo [ ] nombreMetodo (){
return arreglo;
}
```

```java
public static void main(String[] args) {
    int[] arreglo = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    imprimeNumeros(arreglo);
    cambiaNumeros(arreglo);
    imprimeNumeros(arreglo);

}

public static void cambiaNumeros(int[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = array[i] * 2;

    }
}

public static void imprimeNumeros(int[] array) {
    for (int i = 0; i < array.length; i++) {

        System.out.print("valor: " + array[i] + " ");
    }
    System.out.println("");
}
```

Ejercicio

Cree un programa que genere un arreglo de 10 elementos.

Inicialice los elementos con un número aleatorio.

Sume el contenido del arreglo.

Imprima el mayor y el menor de todos los números dentro del arreglo.

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

    int [] arregloAleatorio;

    arregloAleatorio= generaArreglo();
    menorMayorArreglo(arregloAleatorio);
    // TODO code application logic here
}


public static int [] generaArreglo (){

    Random grand = new Random();
    int randomNumber;
    int [] arreglo = new int[10];
    for(int i =0;i<10;i++){
    randomNumber= grand.nextInt(10) + 1;
    arreglo[i]=randomNumber;
        System.out.println("arreglo ["+i+"] : " + arreglo[i]);
    }
return arreglo;
}
```

```java
public static void menorMayorArreglo(int []arreglo){
            int menor=arreglo[0];
            int mayor=arreglo[1];
        for(int i=0 ; i<arreglo.length;i++){
            if(menor>arreglo[i]){
                menor=arreglo[i];
            }
            if(mayor<arreglo[i]){
                mayor=arreglo[i];

            }
        }
    System.out.println("menor " + menor);
    System.out.println("mayor " + mayor);
}
```

Sentencia *for each*

Sintaxis:

```
for (tipo variable: array){
        //sentencias
}
```

Ejemplo

```
for (int n: arrayI){
    System.out.println(n);
}
```

Ordenamiento de burbuja

```
metodoBurbuja (a0,a1,a2, …,a(n-1))
        para i=0 hasta n hacer
            para j=0 hasta n-1 hacer
                si a(j)>a(j+1) entonces
                  cambiarlos de lugar
                fin sin
            fin for
        fin for
fin método
```

Arreglo bidimensionales

- Sintaxis:

  tipo [][] variable;

- Ejemplo

  int [][] arreglo2;

  arreglo2 = new int [3][4];

Recorrido

```
 */
public static void main(String[] args) {

    int[][] arregloBi = {{1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
    {2, 4, 6, 8, 10, 12, 14, 16, 18, 20}};
    System.out.println("Filas= " + arregloBi.length);
    System.out.println("Columnas= " + arregloBi[0].length);
    imprimeNumerosBi(arregloBi);
    cambiaNumerosBi(arregloBi);
    imprimeNumerosBi(arregloBi);

}
```

```
public static void cambiaNumerosBi(int[][] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array[i].length; j++) {

            array[i][j] = array[i][j] * 2;
        }

    }
}

public static void imprimeNumerosBi(int[][] array) {
    for (int[] i : array) {
        for (int j : i) {
            System.out.print("valor: " + j);
        }
        System.out.println("");
    }
    System.out.println("");
}
```

Empaquetado de clases

- Consiste en definir el paquete al cual pertenece la clase.
- Generalmente el nombre del paquete coincide con el nombre del directorio que contiene el archivo .java

Sintaxis

```
package  nombredelpaquete;
```

Consideraciones: la sentencia package es la primer sentencia dentro del archivo.java

Va definida en el import

Afecta a todas las clases dentro del archivo

Modificadores de acceso

- private: Aplicable a atributos y a métodos. Su uso está restringido al interior de la clase.
- (default): Si un atributo o método no define un modificador de acceso, se dice que su acceso es (por default), sólo las clases del mismo paquete tendrá acceso a estos elementos.
- protected: los elementos protected sólo puede ser usado por clases del mismo paquete o por una subclase.
- public: máximo nivel de visibilidad, todos pueden hacer uso de los elementos públicos

Encapsulamiento

```java
package figuras;

public class Rectangulo {
    public int ancho;
    public int alto;

    public int area(){
        return ancho * alto;
    }
}
```

```java
package cbasico;

import figuras.Rectangulo;

public class Cbasico {
    public static void main(String[] args) {
        // TODO code application logic here
        Rectangulo rec = new Rectangulo();
        System.out.println(rec.area());

        rec.alto = 5;
        rec.ancho = 6;
        System.out.println(rec.area());
        rec.alto = 5;
        rec.ancho = -6;
        System.out.println(rec.area());
    }
}
```

Encapsulamiento 2

```java
package figuras;

public class Rectangulo {
    private int ancho;
    private int alto;

    public int area(){
        return ancho * getAlto();
    }

    public void setAncho(int ancho){
        if(ancho>0)
            this.ancho = ancho;
    }

    public int getAncho(){
        return ancho;
    }

    public int getAlto() {
        return alto;
    }

    public void setAlto(int alto) {
        if(alto>0)
            this.alto = alto;
    }
}
```

Sobrecarga de métodos

- Un constructor es un método especial que sirve, generalmente para realizar las tareas que deban realizarse en el momento de crear un objeto. Se invoca al llamar al operador new.
- Características:
  - El nombre del constructor debe ser el mismo que el de la clase.
  - El constructor NO DEBE TENER UN TIPO DE RETORNO.
  - Toda clase debe tener amenos un constructor.
  - Los constructores se pueden sobrecargar,

- Sintaxis:

```
public NombreClase(){

}
```

```
public NombreClase(argumentos){


}
```

```java
public class SobreCarga {
    public void imprime(){
        System.out.println("Cadena por default...");
    }
    public void imprime(String s){
        System.out.println("Valor asignado : "+s);
    }
}
```

```java
public class Cbasico {
    public static void main(String[] args) {
        // TODO code application logic here
        SobreCarga sc = new SobreCarga();

        sc.imprime();
        sc.imprime("Algo");
    }
}
```

Ejercicio

- Defina una clase que implemente un Punto refiriéndose a sus coordenadas cartesianas.
- Implemente el constructor por defecto.
- Implemente el constructor que reciba como argumentos las coordenadas cartesianas.
- Los atributos deben ser enteros.
- Implemente un método que imprima las coordenadas cartesianas con el siguiente formato: “las coordenadas son (x,y)”.

```java
public class Punto{
    private int x;
    private int y;

    Punto(){
    }

    Punto(int x, int y){
        setX(x);
        setY(y);
    }

    public int getX(){
        return x;
    }
    public int getY(){
        return y
    }
    public void setX(int x){
        this.x = x;
    }
    public void setY(int y){
        this.y = y;
    }
}
```

Proyecto 1

- Desarrollar una aplicación que simule a un cajero automático, se debe crear una clase llamada Cuenta, que gestione las operaciones sobre una cuenta bancaria, con los siguientes métodos:

  - void deposito(float c) : Incrementa el saldo de la cuenta.
  - void retiro(float c) : Descuenta saldo de la cuenta.
  - float saldo(): regresa el saldo de la cuenta.

- Por otro lado desarrollar una clase que implemente el main, deberá presentar un menú con las siguientes opciones.

  -crear cuenta vacía.
  -crear cuenta con saldo
  -Realizar un depósito.
  -Realizar un retiro.
  -ver saldo.
  -Salir.

El menú se debe presentar hasta que se presione la opción salir.

Clase String

- La clase String, permite realizar operaciones de manipulación de cadenas. (Cuenta con al menos 15 constructores)
- Sintaxis:

  String variable = new String();

  String variable = new String("Cadena de texto");

  String variable = "Cadena de texto";

El objeto es inmutable, una vez creado, no puede ser modificado.

Algunos métodos de String

- int length(). Regresa el número de caracteres de la cadena.
- boolean equals(String otracadena), true si son iguales las cadenas.
- boolean equalsIgnoreCase(String otra): no verifica si los caracteres son mayúsculas y minúsculas.
- char charAt(int index). Regresa el carácter en la posición Index.
- String toUpperCase(). Regresa una cadena en formato mayúsculas.
- String toLowerCase(). Regresa una cadena en formato minúsculas.
- String substring(int inicio, int final). Regresa un fragmento de la cadena original. Desde inicio - hasta final-1
- String valueOf(Parametro). Regresa una representación en String del objeto parámetro.

Documentación Java SE

http://docs.oracle.com/javase/8/docs/api/index.html

Ejercicio

- Realizar un programa que pida una cadena de teclado.
- El programa deberá imprimir la cadena al revés.

```
 */
public static void main(String[] args) throws ParseException {
    Scanner sc = new Scanner(System.in);
    String cadena = sc.nextLine();
    Cadena ejemploCad = new Cadena(cadena);
    ejemploCad.cadenaAlrevez();
    ejemploCad.cadenaAlrevezBuf();
```

```
/*
esta es un método para la caden al revéz
*/
public void cadenaAlrevez(){
        for (int i = cadena.length()-1; i >= 0; i--) {
                System.out.print(cadena.charAt(i));
    }
        System.out.println("\n");
}

 public void cadenaAlrevezBuf(){
 StringBuffer strB= new StringBuffer();
 strB.append(cadena);
     System.out.println(strB.reverse());
 }

/**
```

Clase System

- Esta clase no puede ser instanciada, provee acceso a atributos que representan las salidas estándar.
- Atributos public:
  - in : Atributo que representa al flujo de entrada estándar, generalmente el teclado.
  - out: Flujo de salida estándar, típicamente corresponde al monitor.
  - err: Flujo de error estándar, típicamente el monitor.

Clase Math

- Clase que proporciona métodos para realizar operaciones matemáticas en un programa, todos los métodos de la clase son de tipo static.
- Constantes:
  - static double E. contiene el valor más cercano al número e.
  - static double PI. contiene el valor más cercano al número pi.
- Algunos métodos.
  - número max(numero a, numero b) regresa el mayor de dos números.
  - número min(numero a, numero b) regresa el menor de dos números.
  - double ceil(double a). Regresa el entero mayor más cercano al número indicado en el parámetro.
  - Double pow(double a, double b), eleva a, a la potencia b

Importación estática

- Consiste en la posibilidad de importar todos los métodos estáticos de una clase de modo que pueda hacerse referencia a los mismos sin necesidad de proporcionar el nombre de la clase.
- Sintaxis

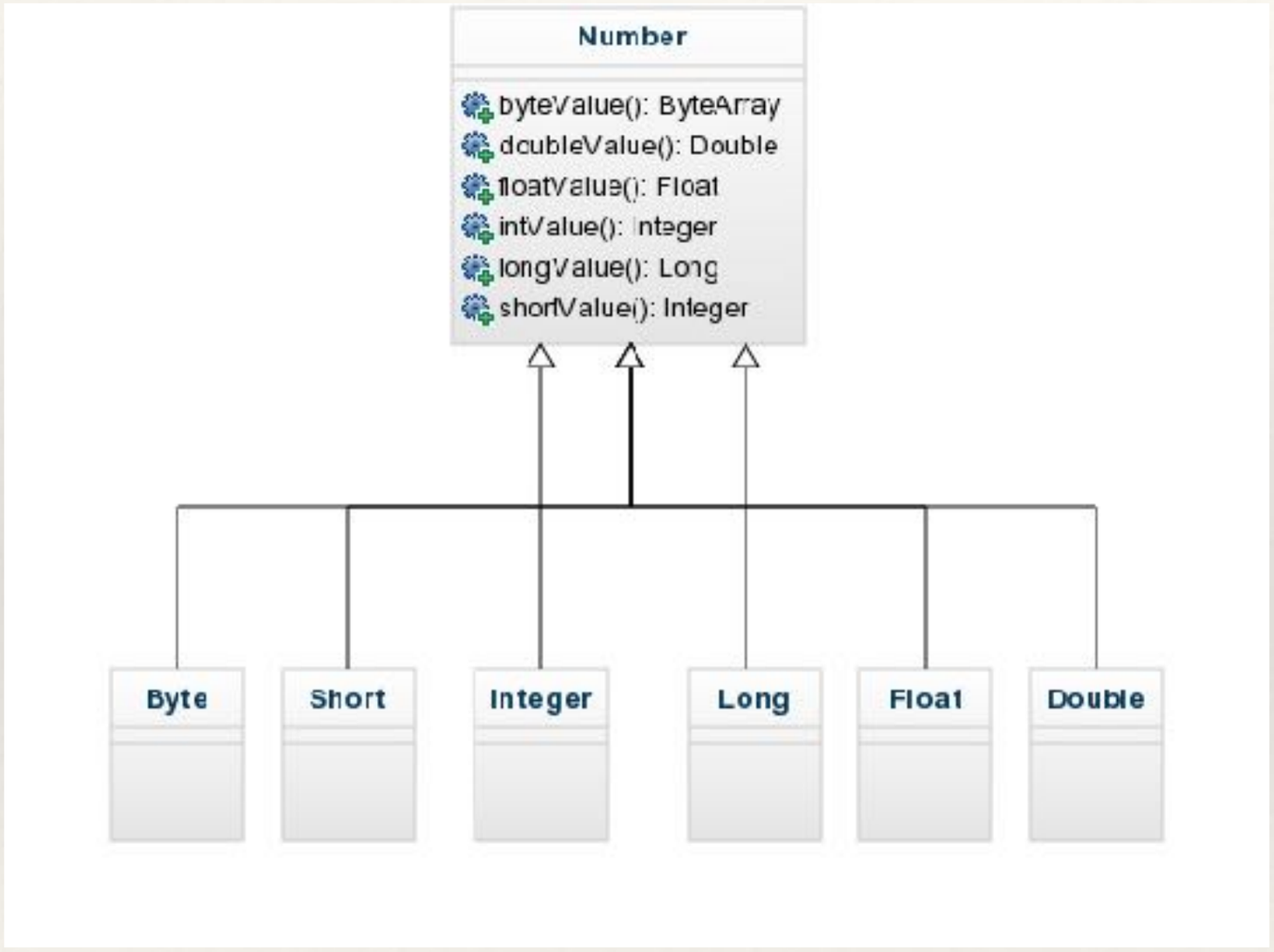
```
import static paquete.clase.*;
```

Ejemplo:

```
import static java.lang.Math.*;
```

```java
public static void metodoString() {

    String cadena = "mensaje";

    System.out.println("caracter " + cadena.charAt(0));
    System.out.println("caracter unicode " + cadena.codePointAt(0))
    System.out.println("contiene XXX " + cadena.contains("XX"));
    String[] subcadenas = cadena.split("s");
    for (String sub : subcadenas) {
        System.out.println("sub " + sub);
    }

}

public static void metodoSystem() {

    System.out.println("salida estandar");
    System.err.println("error!!");

}

public static void metodoMath() {

    System.out.println("PI " + Math.PI);
    System.out.println("E " + Math.E);
    System.out.println("2^3 : " + Math.pow(2, 3));
    System.out.println("max 2,3: " + Math.max(2, 3));
    System.out.println("ceil 2.3 :" + Math.ceil(2.3));
}
```

Clase Wraper



- Encapsulan un objeto primitivo en un objeto
- Conversión de cadenas a tipos primitivos.

Ejemplos

```
int k = 95;

Integer num = new Integer(k);


String s = "3.14";
Float f = new Float(s);

float dato = f.floatValue();
```

Clase Boolean

- Envuelve un dato primitivo tipo boolean.
- Algunos métodos static:
  - parseBoolean(String s): regresa el valor booleano del argumento.
  - compare(boolean a, boolean b); compara dos valores booleanos, si a == b regresa 0; si a es falso y b es verdadero regresa un valor menor a 0 ; si a es verdadero y b es falso regresa un valor mayor a 0

```java
public static void metodoBoolean() {

    boolean x = Boolean.parseBoolean("True");
    int resul = Boolean.compare(true, false);

    System.out.println("X " + x);
    System.out.println("resul " + resul);

}
```

Manejo de fechas: Clase Date

- Pertenece al paquete java.útil
- Representa fecha y hora con precisión de un milisegundo.
- La mayoría de sus métodos tienen estado "Deprecated"

Uso:

```
Date fecha = new Date();

System.out.println(fecha.toString());
```

Clase calendar

- Surge como alternativa a Date para cubrir las carencias de esa clase.
- Es una clase abstracta por lo que no se puede crear un objeto de forma directa.

Uso:

```
Calendar cal = Calendar.getInstance();
```

Algunos métodos:

void set(int año, int mes, int dia): Considerando que los meses inician con base (cero) 0 = January

- void set(int Campo, int valor):

```
cal.set(Calendar.YEAR,2010);
cal.set(Calendar.MONTH, Calendar.JULY);
cal.set(Calendar.DAY, 25);
```

int get(int campo):

```
cal.get(Calendar.YEAR);
cal.get(Calendar.DAY_OF_MONTH);
```

```java
public void metodoCalendar() {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMM dd HH:mm:ss");
    Calendar calendar = new GregorianCalendar(2018, 04, 28);
    System.out.println(sdf.format(calendar.getTime()));
    Calendar calendar2 = new GregorianCalendar(2018, 04, 28, 23, 14, 13);
    int year = calendar2.get(Calendar.YEAR);
    int month = calendar2.get(Calendar.MONTH);
    int day = calendar2.get(Calendar.DAY_OF_MONTH);

    System.out.println("year \t\t: " + year);
    System.out.println("month \t\t: " + month);
    System.out.println("day \t\t: " + day);

    System.out.println(sdf.format(calendar2.getTime()));

}
```

```java
public void metodoDate() throws ParseException {
    SimpleDateFormat sdf = new SimpleDateFormat("dd/M/yyyy");
    String date = sdf.format(new Date());
    System.out.println(date);
    SimpleDateFormat sdf2 = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");
    String dateStr = "27-04-2018 10:20:23";
    Date date2 = sdf2.parse(dateStr);
    System.out.println(date2);

}
```

Clase StringBuilder

- Pensada como un sustituto a la clase StringBuffer, es más rápida en los métodos que implementa.
- Permite adjuntar e insertar información de tipo String, de una forma eficiente.

Sintaxis:

```
StringBuilder sb = new StringBuilder();
```

Uso:

```
sb.append("algo");
sb.insert(3,"lago");
sb.reverse();
```

Clase StringTokenizer

- Clase que permite fragmentar un objeto String en tokens

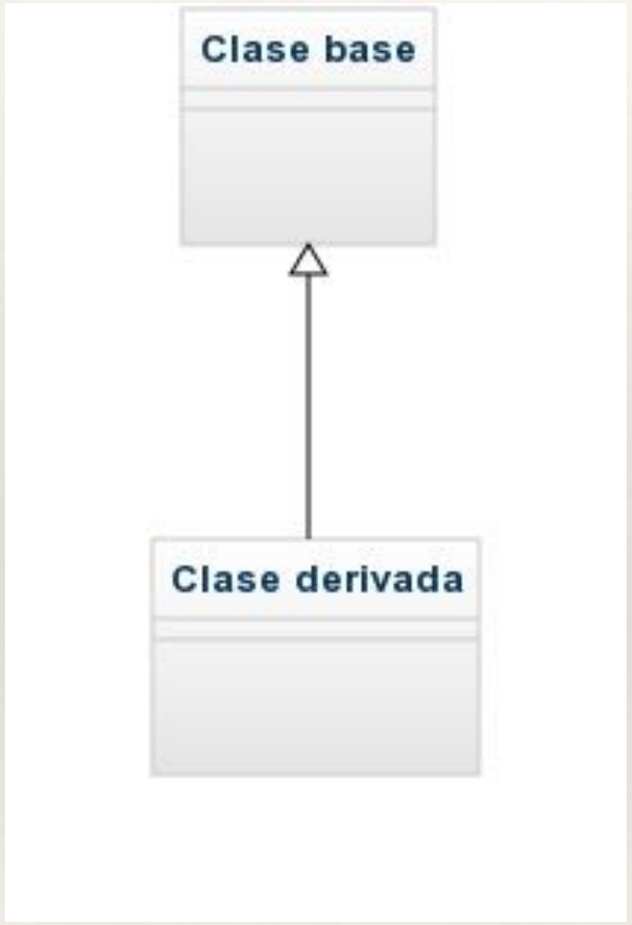
```
StringTokenizer st = new StringTokenizer(“Esto es una prueba");
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

Herencia

- Capacidad de crear clases que adquieran de forma automática los miembros de otras clases existentes, permitiendo añadir atributos y métodos propios.

Ventajas:

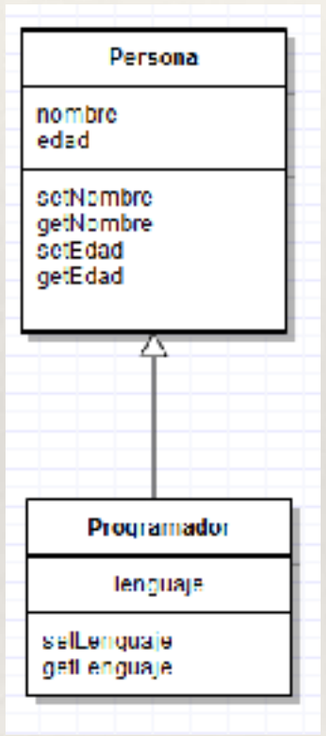
- Facilita la reutilización de código.
- Facilita la mantenibilidad de los proyectos.



Reglas de herencia

1. Java no permite la herencia múltiple
2. Es posible una herencia multinivel
3. Una clase puede ser superclase de varias clases.
4. La clase Object, es la superclase de todas las clases en Java, de forma implícita.

Ejemplo




```java
public class Persona {
    private String nombre;
    private int edad;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

```java
public class Programador extends Persona{
    private String lenguaje;

    public String getLenguaje() {
        return lenguaje;
    }

    public void setLenguaje(String lenguaje) {
        this.lenguaje = lenguaje;
    }

}
```

```java
package herencia;

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

        Programador prog = new Programador();

        prog.setNombre("erandi");
        prog.setEdad(50);
        prog.setLenguaje("Java");


        System.out.println("nombre :"+ prog.getNombre());
        System.out.println("edad : " +prog.getEdad());
        System.out.println("lenguaje : "+prog.getLenguaje());
    }

}
```

Sobre escritura de métodos

```
public void setEdad(int edad) {
    this.edad = edad;
}

public void imprimeDatos(){
    System.out.println(nombre + " : " +edad);
}
```

Método nuevo en Persona.

Método sobre escrito en Programador

```
    this.lenguaje = lenguaje;
}

@Override
public void imprimeDatos(){
    System.out.println(lenguaje);
}
```

```java
package herencia;

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

        Programador prog = new Programador();

        prog.setNombre("erandi");
        prog.setEdad(50);
        prog.setLenguaje("Java");
        prog.imprimeDatos();

    }

}
```

Asignación de objetos derivados a referencias base

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

        Programador prog = new Programador();

        prog.setNombre("erandi");
        prog.setEdad(50);
        prog.setLenguaje("Java");
        prog.imprimeDatos();

        Persona per = new Persona();
        per.setEdad(10);
        per.setNombre("erandi2");
        per.imprimeDatos();

        per=prog;

        per.imprimeDatos();


    }

}
```

Llamada a métodos de la clase base (super)

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

        Programador prog = new Programador();

        prog.setNombre("erandi");
        prog.setEdad(50);
        prog.setLenguaje("Java");
        prog.imprimeDatos();

        Persona per = null;

        per=prog;

        per.imprimeDatos();

    }

}
```

```java
 */
public class Programador extends Persona{
    private String lenguaje;

    public String getLenguaje() {
        return lenguaje;
    }

    public void setLenguaje(String lenguaje) {
        this.lenguaje = lenguaje;
    }

    @Override
    public void imprimeDatos(){
        super.imprimeDatos();
        System.out.println(" : "+lenguaje);
    }
}
```

Ejercicio

- Se tienen tres diferentes vehículos de motor, primero están las motocicletas, los automóviles, los camiones.
- Represente una relación de herencia, definiendo una clase base y las clases derivadas, defina atributos y métodos.
- Implementen las clases y hagan uso de ellas en un main.

Ejecución de constructores en herencia

- Cada que se construye un objeto en Java, primero se ejecuta el constructor de su clase base, y luego el constructor de dicha clase.

- Si se cuenta con una clase con constructores sobrecargados y en la clase derivada se quiere invocar a un constructor de la clase base, se emplea la sentencia "super".

```java
public class Persona {
    private String nombre;
    private int edad;

    public Persona(){
        System.out.println("Constructor de persona");
    }

    public String getNombre() {
        return nombre;
    }
```

```java
 */
public class Programador extends Persona{
    private String lenguaje;

    public Programador(){
        System.out.println("constructor de programador");
    }

    public String getLenguaje() {
        return lenguaje;
    }
```

Clases *final*

- Denominadas clases finales, son clases que no permiten realizar herencia a partir de ellas.
- Se definen finales asignando el modificador *final*, antes que la definición class.

Sintaxis

```
public final class ClaseA{
// implementación de la clase.
}
```

Ejercicio

- Utilicen sus ejercicios de vehículos de motor, para implementar la llamada a constructores o métodos de la propia clase base, en las clases derivadas.
- Definan una clase derivada como *final*, intenten realizar herencia de esa clase.

Clases *abstractas*

- Es una clase en la cual, alguno de sus métodos esta declarado como método abstracto, pero no está implementado.
- Un método se define como abstracto porque no se hará su implementación, de eso se encargarán las clases derivadas.

Sintaxis:

```
public abstract class NombreClase{
        public abstract  tipo_retorno nombreMetodo(argumentos);
        // otros métodos
}
```

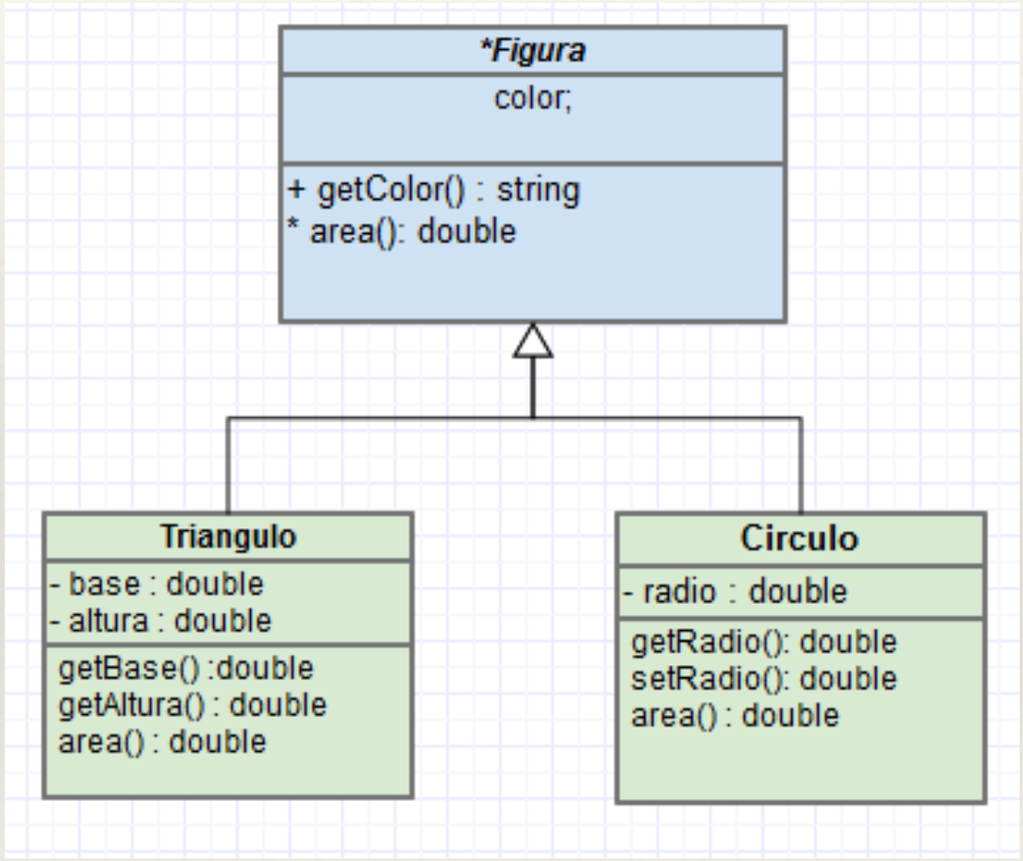
Aspectos importantes

- Una clase abstracta puede tener métodos NO abstractos.
- No es posible crear objetos de una clase abstracta.
- Las sub-clases de una clase obstracta, están obligadas a sobreescribir los métodos abstractos de su superclase.
- Una clase abstracta puede tener constructores.

```
public abstract class Figura{
        public abstract double  area();
        ….
}
```

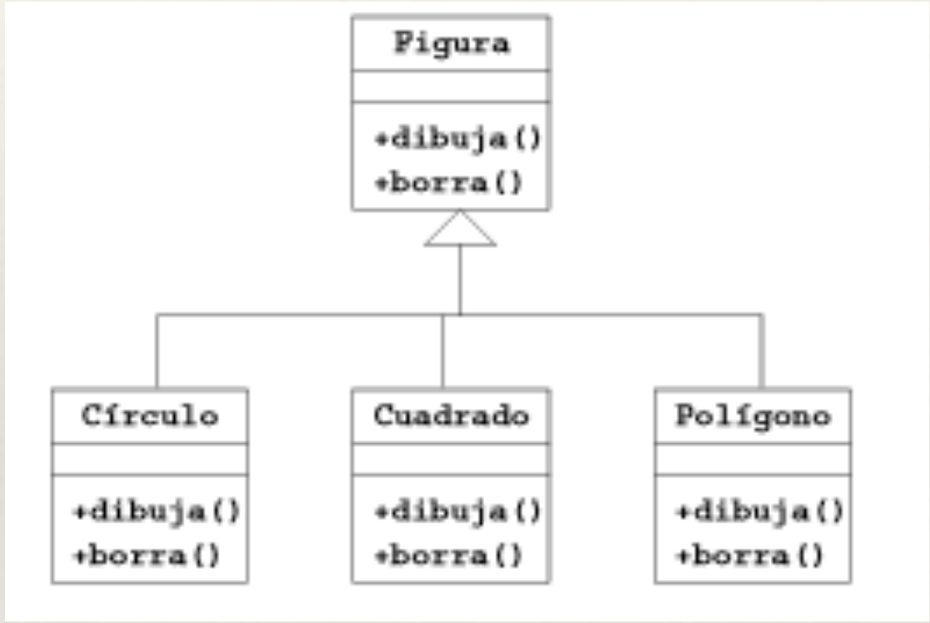
Ejercicios

Implemente el diagrama de clases que se muestra, la clase figura es una clase abstracta.



Polimorfismo

- La palabra implica "Múltiples formas"
- Las operaciones polimórficas son aquellas que hacen funciones similares con objetos diferentes.



- Es posible tener una operación polimórfica pintarFiguras() que opere correctamente con cualquier clase de figura:

pintarFigura

borrar

dibujar figura

Esta operación debería :

- Llamar a la operación borrar de la figura que se trate. Si es un círculo, o un cuadrado.

```java
public class ClaseFigura {
    public void dibuja(){
        System.out.println("dibuja figura");
    }

    public void borra(){
        System.out.println("borra figura");
    }

    public void pintarFigura(){

        borra();
        dibuja();
    }

}
```

```java
public class Cuadrado extends ClaseFigura {
    @Override
    public void dibuja(){
        System.out.println("dibuja cuadrado");
    }

    @Override
    public void borra(){
        System.out.println("borra cuadrado");
    }

}
```

```java
public static void main(String[] args) {
    ClaseFigura fig = new ClaseFigura();
    fig.pintarFigura();

    Cuadrado cua = new Cuadrado();
    cua.pintarFigura();


}
```

Propiedades del polimorfismo

- Una referencia de una superclase puede apuntar a un objeto de cualquiera de sus subclases.

```
Figura  f1  = new Circulo();

Figura f2 = new Rectangulo();
```

- La operación se selecciona en base a la clase del objeto, no a la referencia.

- Estas propiedades permiten tener la siguiente función;

```
public void pintaFigura(Figura f){
        f.borra();
        f.dibuja();
}
```

Podría invocarse de la siguiente manera.

```
Circulo c = new Circulo(…);
Triangulo t = new Triangulo(…);
pintaFigura(c);
pintaFigura(p);
```

```java
public static void main(String[] args) {
    ClaseFigura fig = new ClaseFigura();
    fig.pintarFigura();

     Cuadrado cua = new Cuadrado();
    cua.pintarFigura();

     Circulo c = new Circulo();
    Poligono p = new Poligono();
    pintar(c);
    pintar(p);


}

 public static void pintar(ClaseFigura f){
            f.borra();
            f.dibuja();
    }
```

Algunas restricciones

- Java permite que una referencia a una superclase pueda referenciar a un objeto de sus subclases, pero no a la inversa.

```
Figura f =  new Triangulo(…);  // válido.

Triangulo t = new Figura(…);  // No permitido.
```

Conversión de referencias: casting

- Es posible convertir referencias.

```
Persona p = new Programador();

Programador prog = (Programador) p;
```

Es casting cambia el punto de vista con el que vemos al objeto.

- Una conversión de tipos incorrecta produce una excepción de tipo ClassCastException en tiempo de ejecución.

```
Figura f = new Figura();
Triangulo t = (Triangulo) f;
```

Es necesario verificar el tipo de instancia usando instanceof.

Array de objetos

- Debido al polimorfismo es posible que un array contenga referencias a objetos de distintas clases.
  - Superclase y todas sus subclases.

Ejemplo:

```
        Figura[] figuras = new Figura[3];
        figuras[0]= new Rectangulo();
        figuras[1]= new Circulo();
        figuras[2] = new Poligono();

for (int i =0 ; i<figuras.length ; i++){
        figuras[i].borrar();
}
```

Interfaces

- Va más allá de una clase abstracta, es una clase donde todos los métodos son abstractos, permite al desarrollador establecer el comportamiento que tendrán las clases que implementen la interfaz.

Sintaxis:

```
interface NombreInterfaz {
    tiporetorno nombreMetodo(argumentos);
}
```

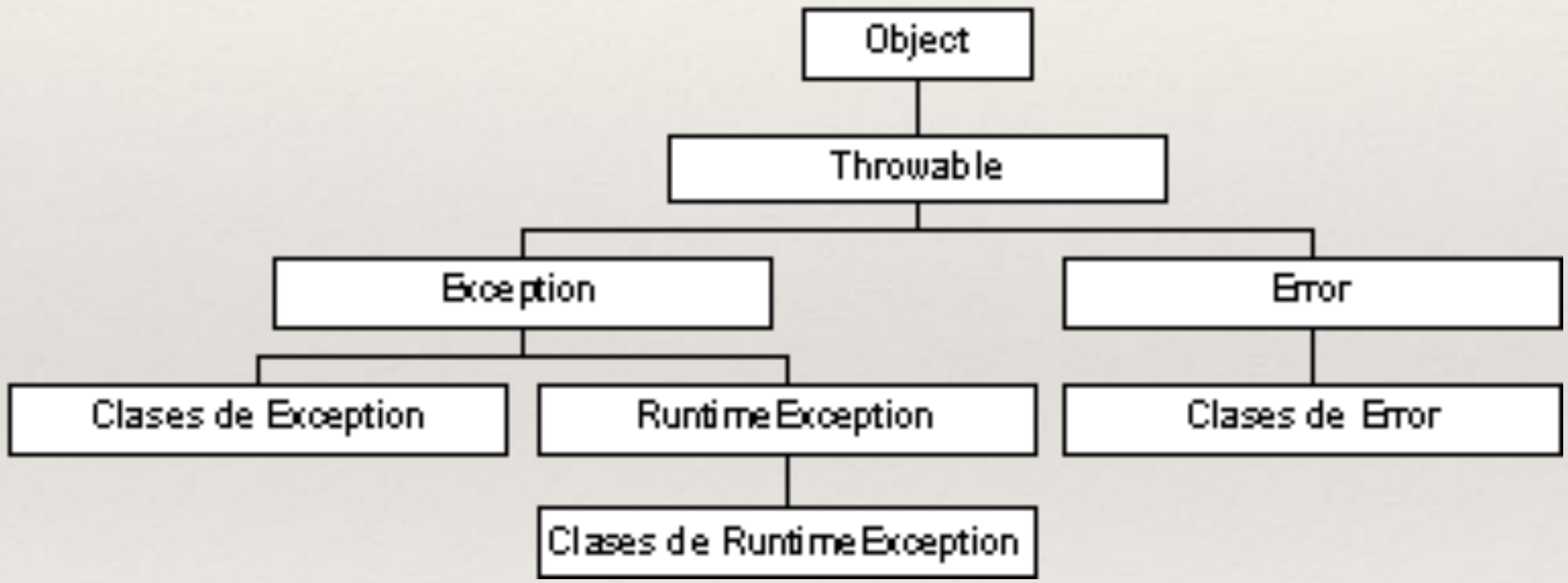
Interface vs clase abstracta

- Interfaz es una lista de métodos no implementados, puede incluir declaración de constantes.
- Clase abstracta puede incluir métodos implementados y no implementados o abstractos, así como atributos.

Tipo de excepciones

- *Error*: Excepciones que indican problemas muy graves, que suelen ser no recuperables y no deben casi nunca ser capturadas.
- *Exception*: Excepciones no definitivas, pero que se detectan fuera del tiempo de ejecución.
- *RuntimeException*: Excepciones que se dan durante la ejecución del programa.

Manejo de excepciones



- Para que el sistema de gestión de excepciones funcione, se ha de trabajar en dos partes de los programas:
  - Definir qué partes de los programas crean una excepción y bajo qué condiciones. Para ello se utilizan las palabras reservadas *throw* y *throws*.
  - Comprobar en ciertas partes de los programas si una excepción se ha producido, y actuar en consecuencia. Para ello se utilizan las palabras reservadas *try, catch* y *finally*.

Sintaxis

```
try{
        // sentencias que posiblemente generen excepciones.
}catch( tipo_de_excepción ){
        // código para solucionar la excepción.
}
```