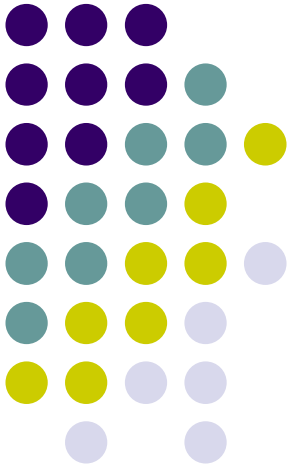
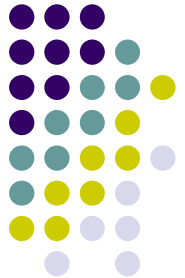
Java



Objetos

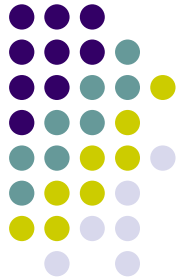
Lic. Claudio Zamoszczyk
claudio@honou.com.ar

Contenidos



- Setter, Getters,This, Super, Constructores
- Paquetes
- Herencia
- Polimorfismo
- Arrays

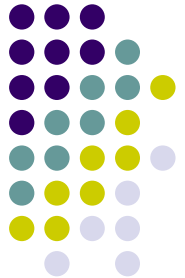
Programación Orientación a Objetos: Getter y Setter



- **Permiten acceder a variables que poseen visibilidad reducida (Private), fomentando el ocultamiento de la información y las buenas prácticas de programación Java.**

```
public class Animal{
        private int edad;

        public getEdad() {
                return edad;
         }

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

}

Animal perro = new Animal();
perro.setEdad(3);
System.out.println(“La edad del perro es: ” + perro.getEdad());
```

Programación Orientación a Objetos: this

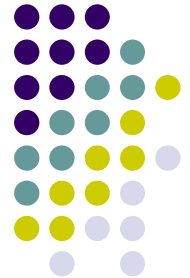


- **Es una variable especial de sólo lectura que proporciona Java. Contiene una referencia al objeto en el que se usa dicha variable. A veces es útil que un objeto pueda referenciarse a si mismo.**

- **public void asignarSaldo(double saldo) {**

```
    //this.saldo = variable del objeto Cuenta
    //saldo = variable definida sólo dentro del metodo
    this.saldo =saldo;
}
```

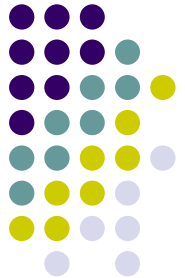
Programación Orientación a Objetos: Paquetes (Package)



- **Un paquete (package) es una agrupación de clases. Lo habitual es agrupar un conjunto de clases que tienen alguna relación lógica dentro de un mismo paquete.**

- **Los paquetes pueden ser creados por el programador o utilizar el gran número que existe en las APIs de Java.**

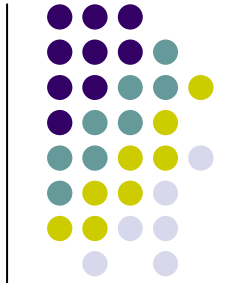
Programación Orientación a Objetos: Paquetes (Package)



```
package animales;

public class Perro {
      private String nombre;
      ….
}
```

```
package negocios;

import animales.*;

public class VentaDeMascotas {
      private String nombre;
      ….
}
```

Programación Orientación a Objetos: Contructores

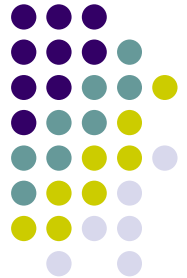


- **Los Constructores son métodos cuyo nombre coincide con el nombre de la clase y que nunca devuelven ningún tipo de dato, no siendo necesario indicar que el tipo de dato devuelto es void.**

- **Los constructores se emplean para inicializar los valores de los objetos y realizar las operaciones que sean necesarias para la generación de este objeto (crear otros objetos que puedan estar contenidos dentro de este objeto, abrir un archivo o una conexión de internet, etc).**

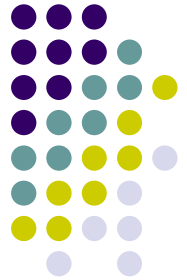
- **Como cualquier método, un constructor admite sobrecarga.**

Programación Orientación a Objetos: Contructores



```
public class Animal{
        private int edad;
        private String nombre;

        public Animal() {
                nombre = “no tiene por el momento”;
        }

        public Animal(String nombre){
                this.nombre = nombre;
        }

        public Animal(String nombre, int edad){
                this.edad = edad;
                this.nombre = nombre;
        }

        public void nace(){
                System.out.println("Hola mundo");
        }

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

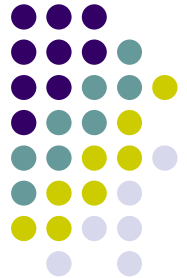
Programación Orientación a Objetos: Herencia



- **Cuando en Java indicamos que una clase “extends” otra clase estamos indicando que es una clase hija de esta, hereda todos sus métodos y variables.**

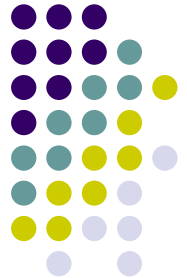
- **Este es un poderoso mecanismo para la reusabilidad del código. Podemos heredar de una clase, por lo cual partimos de su estructura de variables y métodos, y luego añadir lo que necesitemos o modificar lo que no se adapte a nuestros requerimientos.**

Programación Orientación a Objetos: Herencia



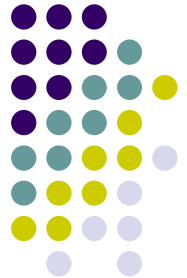
```
public class Animal{
      private int edad;
      private String nombre;


      public String getNombre(){
            return nombre;
      }
……..

public class Perror extends Animal {

      public void ladrar() {
            …….
      }

}
```

Programación Orientación a Objetos: super



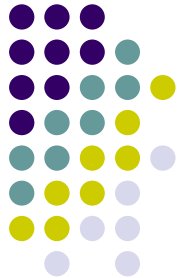
- **Del mismo modo que this apunta al objeto actual tenemos otra variable super que apunta a la clase de la cual hereda (extiende) nuestra clase**

Programación Orientación a Objetos: super



```
public class Gato {
        public void hablar(){
                System.out.println("Miau");
        }
}

public class GatoMagico extends Gato {
        private boolean gentePresente;

        public void hablar(){
                if(gentePresente) {
                //Invoca al método sobreescrito de la clase padre
                  super.hablar();
                } else {
                  System.out.println("Hola");
                }
        }
}
```

Programación Orientación a Objetos: clases abstractas

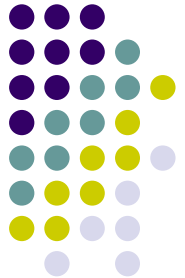


- **Una clase abstracta (abstract) es una clase de la que no se pueden crear Objetos. Su utilidad es permitir que otras clases deriven de ella, proporcionándoles un marco o modelo que deben seguir y algunos métodos de utilidad general.**

- **Una clase abstracta puede tener métodos declarados como abstract, en cuyo caso no se da definición del la lógica de programación del método.**

- **Cualquier clase hija que extienda de una clase abstracta esta obligada a implementar los métodos de su padre que sean abstractos.**

Programación Orientación a Objetos: clases abstractas



- **public abstract Animal {**

  …

  public abstract void caminar();

public Perro extends Animal {

…

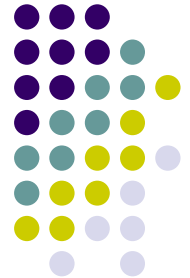
  public void caminar() {

    System.out.println(“Yo camino en 4 patas”);

  }

….

Programación Orientación a Objetos: clases y métodos finales



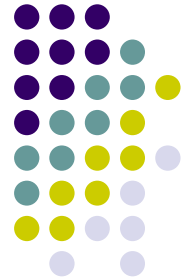
- **Cuando declarábamos variables como final su valor no se podía cambiar durante toda la ejecución de a aplicación.**

- **Final aparte de aplicarse a variables también funciona al declarar final a una clase y a un método.**

- **En el primer caso, declarando a una clase como final, impedimos que esta pueda ser heredada por otras clases.**

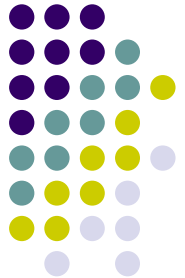
- **Análogamente un método declarado como final no puede ser redefinido “sobrescrito” por otra clase que herede su comportamiento.**

Programación Orientación a Objetos: clases y métodos finales



```
public final class ManejadorDeConexiones {
    …..
}

public class Animal {
    public final respirar() {
        …….
    }
    …..
}
```

Programación Orientación a Objetos: Casting

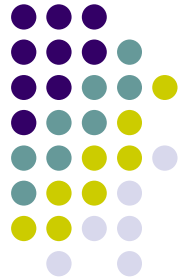


- **El casting es un procedimiento para transformar un objeto de una clase a otra clase siempre y cuando haya una relación de herencia entre ambas-**

- **Implícito: no se necesita escribir código para que se lleve a cabo. Ocurre cuando se realiza una conversión ancha (widening casting), es decir, cuando se coloca un valor pequeño en un contenedor grande.**

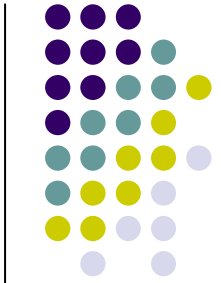
- **Explícito: sí es necesario escribir código. Ocurre cuando se realiza una conversión estrecha (narrowing casting), es decir, cuando se coloca un valor grande en un contenedor pequeño.**

Programación Orientación a Objetos: Casting



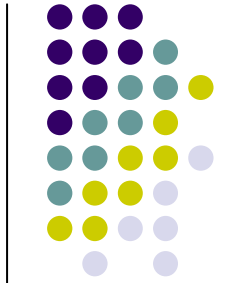
```
public Perro extends Animal {
…
}

Perro perroLindo = new Perro();
Animal animalX = perroLindo;
Perro perroX = (Perro) animalX;
```

Programación Orientación a Objetos: Interfaces

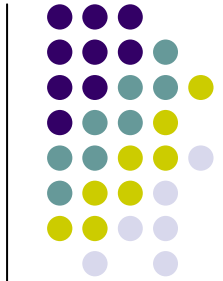


- **Un interface es parecido a una clase abstracta en Java, pero con las siguientes diferencias:**
- **- Todo método es abstracto y público sin necesidad de declararlo. Por lo tanto un interface en Java no implementa ninguno de los métodos que declara.**
- **- Un interface se implementa (implements) no se extiende (extends) por sus clases hijas.**
- **- Una clase puede implementar más de un interfaz en Java, pero sólo puede extender una clase. Es lo más parecido que tiene Java a la herencia múltiple.**

Programación Orientación a Objetos: Interfaces



```
public interface Figura{
      public int area();
}

public class Cuadrado implements Figura {

      private int lado;
      public Cuadrado (int ladoParametro) {
            lado = ladoParametro;
      }
      public int area(){ return lado*lado; }
}


public class PruebaInterfaz{

public static void main(String args[]) {
      Figura figura=new Cuadrado (5);
      System.out.println(figura.area());
}
```

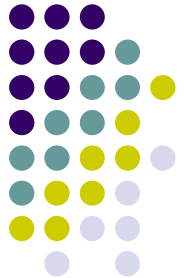


Ejercicios

1 – Crear una clase Persona con el atributo nombre, luego crear un clase Empleado y otra Gerente que extiendan de persona. Imprimir por pantalla el nombre de 1 empleado y 1 Gerente pero en ambos casos tratarlos como Personas.

Nota: Utilizar una clase UtilsPersonas que tenga un método para imprimir nombre de los objetos persona.

Arrays



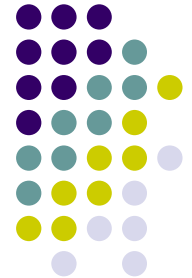
En Java los arrays son un objeto. Como tales se crean mediante el comando new. La sintaxis en la definición de un array es la siguiente:

TipoDatos[] nombreArray = new TipoDatos[tamaño_array];

TipoDatos es el tipo de los datos que se almacenarán en el array (int, char, String... o cualquier objeto). TamañoArray es tamaño que le queremos dar a este array.

Int[] edades = new int[10];

Arrays



Para crear un array en Java hay dos métodos básicos. Crear un array vacío:

```
int lista[] = new int[50];
```

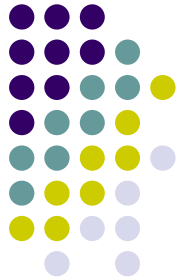
o se puede crear ya el array con sus valores iniciales:

```
String nombres[] = { "Juan","Pepe","Pedro","Maria" };
```

Esto que es equivalente a:

```
String nombres[];
nombres = new String[4];
nombres[0] = new String("Juan");
nombres[1] = new String("Pepe");
nombres[2] = new String("Pedro");
nombres[3] = new String("Maria");

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

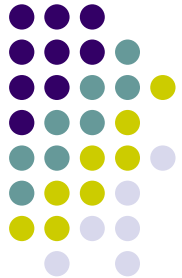
Ejercicios



1 - Crear una matriz unidad de 4 x 4 e imprimir su resultado por pantalla.

1.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

Ejercicios



2 - Cargar por teclado 3 nombres, generar objetos Persona con los nombres correspondientes, colocar los objetos dentro de un Array y luego imprimirlos por consola

Ingrese el nombre para el objeto perdona:
……
,,,,,,,

Juna
Carlos
Maria

¿Preguntas?