Conceptos fundamentales de Java
7-5: Polimorfismo
Actividades practices

Objetivos de la lección:

- Aplicar referencias superclase a subclase objetos
- Redactar un código para sobrescribir métodos
- Usar el enlace dinámico para soportar el polimorfismo
- Crear métodos y clases abstract
- Reconocer el proceso de sobrescritura de método correcto
- Usar el modificador final
- Explicar el prototipo y la importancia de la clase Objeto
- Redactar un código para un applet que muestra dos triángulos de diferentes colores
- Describir las referencias a objetos

Vocabulario:

Identifique el término correspondiente a cada una de las siguientes definiciones.

	Un concepto en la programación orientada por objetos que les permite a las clases tener varias formas y comportarse como sus superclases.
	Implementar métodos en una subclase que tiene el mismo prototipo (los mismos parámetros, nombre de método, y tipo de retorno) que otro método en la superclase.
	Una palabra clave en Java usada para limitar las subclases a partir de la extensión de una clase, sobrescribiendo métodos o cambiando datos.
	Una propiedad de una case estática que hace que la clase no pueda ser extendida o que sus datos sean cambiados.
	Implementar un método con el mismo nombre que otro método en la misma clase que tiene diferentes parámetros o un tipo de retorno diferente.
	El proceso por el cual Java es capaz de determinar qué método invocar cuando los métodos han sido sobrescritos.
	Una palabra clave en Java que permite que las clases sean extendidas, pero las clases no pueden ser instanciadas (construidas)y cuando se aplican los métodos, se indica que los métodos deberían ser implementados en todas las subclases de la clase.

Inténtelo/resuélvalo:

1. ¿Cuál sería el resultado del siguiente código?

```
class A
{
    void callthis() {
        System.out.println("Inside Class A's Method!");
    }
}

class B extends A
{
    void callthis() {
        System.out.println("Inside Class B's Method!");
    }
}

class C extends A
{
    void callthis() {
        System.out.println("Inside Class C's Method!");
    }
}

class DynamicDispatch {
    public static void main(String args[])  {
        A a = new A();
        B b = new B();
        C c = new C();
        A ref;

        ref = b;
        ref.callthis();

        ref = c;
        ref.callthis();

        ref = a;
        ref.callthis();
    }
}
```

2. ¿Cuál es la diferencia entre una clase abstract y una interfaz? ¿Cuándo es apropiado el uso de una clase abstract o una interfaz?

3. Dada la siguiente información, determine si resultará: “siempre compila”, “a veces compila”, “no compila”.

    public interface A

    public class B implements A

    public abstract class C

    public class D extends C

    public class E extends B

    Cada clase ha sido inicializada, pero no queda claro para qué han sido inicializadas:

        A a = new...

        B b = new...

        C c = new...

        D d = new...

        E e = new...

    Se incluyen los siguientes métodos:

    interfaz A especifica method void methodA()

    clase C tiene el abstract method void methodC()

Código:	¿Siempre compila, a veces compila, no compila?
a = new B();	
d = new C();	
b.methodA();	
e.methodA();	
c = new C();	
(D)c.methodC();	

4. Sobrescribir el método toString() para la clase más abajo para generar los resultados, haciendo coincidir con el resultado dado. El método toString() debería imprimir todos los valores desde 1 al número especificado en num y luego imprimir el valor final usando el método getFactorial provisto.

    Suponga que el num int variable es un valor global público:

    “Factorial: 10! = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 = 3628800”

```
int getFactorial(){
   int factorial;
       for(i = num; num > 0; i--){
              factorial *= num;
       }
       return factorial;
      }
   public String toString() {

   }
```