

Autores: Álvaro Quesada Pimentel y Daniel Serrano Torres.

On the use of User Interfaces and/or commands in the Cycling Problem.

En este documento se explica en que consiste la práctica que ha realizado, su alcance, sus objetivos, las soluciones dadas, etcétera…

Tecnología de la programación:  
Practica dos.

Índice

[1. Requisitos 1](#_Toc341342821)

[2. Objetivos 1](#_Toc341342822)

[3. Resumen 1](#_Toc341342823)

[4. Abstract 1](#_Toc341342824)

[5. Observaciones 1](#_Toc341342825)

[6. Sugerencias 1](#_Toc341342826)

[7. Introducción 1](#_Toc341342827)

[8. Objetivos que se alcanzarán 1](#_Toc341342828)

[9. Relación con la docencia cursada 1](#_Toc341342829)

[10. Viabilidad 1](#_Toc341342830)

[11. Estado del arte y fundamentación teórica 1](#_Toc341342831)

[12. Conclusión 1](#_Toc341342832)

[13. Bibliografía 1](#_Toc341342833)

**PRACTICA UNO**

On the use of User Interfaces and/or commands in the Cycling

**Fecha de entrega**: 10 de Enero de 2013

1. **Requisitos**

Para realizar esta práctica es preciso conocer el temario impartido en clase, comprendido en los temas primero y quinto.

Este documento se va a centrar en explicar las diferencias entre clases y objetos, la herencia y sus relaciones y tipos.

**¿Qué es una clase?**

Es un conjunto coherente que consiste en un tipo particular de metadatos (información sobre los datos, son datos que describen otros datos y nos permite obtenerlos).

Es un modelo que describe el estado y el comportamiento que tienen todos los objetos del mundo real.

Encapsula el estado y el comportamiento del concepto que representa, encapsula los atributos y comportamientos.

Una clase es una plantilla o un prototipo para crear objetos, por eso se dice que los objetos son instancias de clases.

**¿Qué es un objeto?**

Es el encapsulamiento de un conjunto de operaciones (métodos) que pueden ser invocados externamente, y de un estado que recuerda el efecto de los servicios.

Propiedades de un objeto, el tiempo de vida, estado, y el comportamiento definido por los métodos.

**Herencia**

En orientación a objetos la herencia es, después de la agregación o composición, el mecanismo más utilizado para alcanzar objetivos como lo son la reutilización y la extensibilidad. A través de ella los diseñadores pueden crear nuevas clases partiendo de una clase o de una jerarquía de clases preexistente evitando con ello el rediseño, la modificación y verificación de la parte ya implementada. La herencia facilita la creación de objetos a partir de otros ya existentes e implica que una subclase obtiene todo el comportamiento y eventualmente los atributos de su superclase.

**Tipos de herencia**

En el caso del lenguaje de programación escogido para está práctica y utilizado en esta cátedra existen dos tipos de herencia.

El primer tipo de herencia es la herencia de estructura de datos y comportamiento, donde una clase hija obtiene todo lo que su padre le proporciona, tanto datos como comportamiento.

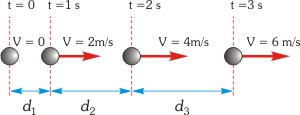
El segundo tipo de herencia, es la herencia de comportamiento o de interfaz, la cual se aplica con interfaces, esta herencia permite definir el comportamiento que se va a heredar por los elementos hijos.

En java la herencia múltiple solo se puede realizar con la herencia de interfaz o de comportamiento.

También será necesario tener un conocimiento básico de física, relacionado con el movimiento rectilíneo uniformemente variado. Se denomina así a aquel movimiento rectilíneo que se caracteriza porque su aceleración **a** permanece constante en el tiempo (en módulo y dirección).

En este tipo de movimiento el valor de la velocidad aumenta o disminuye uniformemente al transcurrir el tiempo, esto quiere decir que los cambios de velocidad son proporcionales al tiempo transcurrido, o, lo que es equivalente, en tiempos iguales la velocidad del móvil aumenta o disminuye en una misma cantidad.

Veamos un ejemplo:



En este caso tenemos un móvil que se mueve horizontalmente describiendo un MRUV en donde en cada segundo el valor de su velocidad aumenta en 2 m/s. Debido a esto, el valor de la aceleración constante con que se mueve el móvil es 2 metros por segundo cuadrado:

**a = 2 m/s2**

1. **Objetivos**
   1. Generales

El objetivo de la práctica es trabajar con la herencia, una característica de la programación orientada a objetos. Tratando de crear un sistema que represente un ciclista.

Además se pide la incorporación de un parser que traduzca unos comandos introducidos por teclado o fichero a unos órdenes que pueda entender el ciclista.

También se introducirá una carretera por la que ira la bici, esta carretera estará dividida en tramos, y estos tramos tendrán una pendiente, positiva o negativa, y un viento que puede ser a favor o en contra.

Para ello se recurrirá a los conocimientos sobre polimorfismo, sobrecarga, sobrescrita, vinculación estática y dinámica. Se utilizarán en la medida que sean necesarios.

* 1. Particulares
* Estudiar la herencia simple y múltiple en la programación orientada a objetos, y la herencia de interfaz.
* Construir un manejador del tiempo como contenedor de números. (horas, minutos y segundos).
* Construir un paseador de ordenes sencillo que permita la interacción del

Usuario con el ciclista.

* Construir un mapa de carretera en el que se especifique la pendiente y el viento de los tramos.
* Recordar conceptos de física básicos (movimiento rectilíneo uniformemente variado).
* Utilizar eclipse como entorno de desarrollo y jUnit (librería de java para realizar pruebas sobre el sotfware).

1. **Resumen**

Se ha realizado una simulación de un ciclista y su bicicleta con el fin de incluirlo en posteriores ampliaciones. Se ha considerado que el ciclista es una persona y que existen numerosas "apariencias" o "vistas" de bicicleta. El tiempo se ha acumulado en un reloj y se ha tratado para permitir nuestro interés. La salida de datos se ha considerado parte fundamental del problema para poder apreciar el funcionamiento de la simulación. Se ha hecho una validación y verificación exhaustiva.

El ciclista va a recibir una serie de órdenes. Estas órdenes indicaran al ciclista lo que debe hacer, por ejemplo frenar, acelerar, aumentar plato, etc. Se transmitirán al ciclista por medio de teclado o de un fichero.

También se le incorporara al ciclista una carrera por la que ira circulando, esta carretera estará dividida en tramos, y estos tramos tendrán la pendiente y el viento que hay en ese punto.

1. **Abstract**

There has been a simulation of a cyclist and his bike in order to include it in subsequent extensions. It has been considered that the rider is a person and that there are many "appearances" or "views" of cycling. The time is accumulated in a clock and has tried to allow our interest. The data output is considered part of the problem in order to assess the performance of the simulation. It has made a thorough validation and verification.

1. **Observaciones**

El proceso de diseñar el sistema de la salida de datos has sido ligeramente costoso, ya que la manera correcta de hacerlo es una representación antinatural del “mundo real” el cual tratamos de representar en nuestro modelo de software.

La creación del parser ha sido bastante dura, puesto que el paso de comandos a ordenes puede llegar a ser un proceso costoso y de difícil aprendizaje.

1. **Sugerencias**

Definir los objetivos de las prácticas más exhaustivamente y proporcionar el material utilizado en las sesiones.

1. **Introducción**

El estudio de la herencia en POO con el uso de la herramienta Java presenta restricciones que no aparecen en otros lenguajes como C++. Es el caso de la herencia múltiple, que en java se resuelve con la herencia de interface.

El hecho de que dichas restricciones o mecanismos tengan palabras clave en Java para estos conceptos indica que estas ideas fueron consideradas los suficientemente importantes como para proporcionar un soporte directo. La palabra clave **interface** lleva el concepto de abstracción un paso más allá. L a palabra clave abstract permite crear uno o más métodos no definidos dentro de una clase: proporcionamos parte de la interfaz, pero sin proporcionar la implementación correspondiente. La implementación se proporciona de las clases que hereden de la clase actual. La palabra clave **interface** produce una clase completamente abstracta, que no proporciona ninguna implementación en absoluto. Las interfaces permiten al creador determinar los nombres de los métodos, las listas de argumentos y los tipos de retorno, pero sin especificar ningún cuerpo de ningún método. Una interfaz proporciona simplemente un comportamiento o forma, sin ninguna implementación.

El modelo elegido ha sido una aplicación que tiene una persona, que es un ciclista, y este a su vez monta una bicicleta, y lleva un "reloj" que mide su propio tiempo de forma habitual. Este modelo muestra todos los datos del modelo por la Salida de Datos.

Atendiendo al desacoplamiento completo. Cuando un método funciona con una clase en lugar de con una interfaz, estamos limitados a utilizar dicha clase o sus subclases. Si quisiéramos aplicar ese método a una clase que no se encontrara en esa jerarquía, no podíamos. Las interfaces relajan esta restricción considerablemente. Como resultado, permiten escribir código más reutilizable.

1. Introducción a la práctica

La aplicación es capaz de dejar correr de manera secuencial todos los objetos durante el transcurso de un determinado tiempo de ejecución, mostrando su estado en cada momento.

1. Conceptos y definiciones

Herencia simple: Se produce cuando una clase extiende o hereda únicamente de otra o de una sola interfaz.

Herencia múltiple: Una clase puede heredar las características de varias clases base, es decir, puede tener varios padres. En este aspecto hay discrepancias entre los diseñadores de lenguajes. Algunos de ellos han preferido no admitir la herencia múltiple debido a que los potenciales conflictos entre métodos y variables con igual nombre, y eventualmente con comportamientos diferentes crea un desajuste cognitivo que va en contra de los principio de la programación orientada a objetos. Por ello, la mayoría de los lenguajes orientados a objetos admite herencia simple.

Herencia de comportamiento o interfaz: Este tipo de herencia es característico del lenguaje Java, donde se utiliza para realizar la herencia múltiple.

1. **Objetivos que se alcanzarán**

* Comprender y utilizar los diferentes tipos de herencia, las vinculaciones dinámicas y estáticas.
* Aprender a documentar una práctica para futuros proyectos.
* Aprender el mecanismo por el cual una serie de comandos se convierten en unas ordenes que pueda interpretar nuestra aplicación.

1. **Relación con la docencia cursada**

* Esta práctica nos introduce a la herencia, el polimorfismo, etc...
* En ella van a estar involucrados los conocimientos adquiridos sobre la programación orientada a objetos, herencia, polimorfismo, vinculación dinámica y estática, abstracción de un problema real, etc..

El modelado de clases, la utilización de objetos como elementos concretos del problema. La interacción de estos elementos por el sistema de paso de mensajes que caracteriza a la programación orientada a objetos.

1. **Viabilidad**

Esta práctica resulta viable dado que su envergadura y complejidad es menor, si bien todos los conceptos y conocimientos necesarios para realizarla eran nuevos y ha sido necesario que comprenderlos y conseguir utilizarlos correctamente.

Pero desde el punto de vista de los conocimientos nuevos que han estado involucrados en la realización de la práctica, se ha vuelto ligeramente complicado realizar el modelo del sistema que se ha construido correctamente.

1. **Estado del arte y fundamentación teórica**

En esta práctica nos hemos encontrado con varios problemas. Estos problemas se citan a continuación con su correspondiente solución elegida.

**1º Problema**

A la hora de realizar el modelo del sistema, sobre todo en cuanto a la estructura de la herencia, es preciso conocer el porqué del la existencia de los tipos de herencia en el lenguaje Java.

* **Antecedente**

La herencia en la programación orientada a objetos es algo habitual y fundamental, ya que con ella se alcanzan algunos de los objetivos más preciados en el desarrollo del software como son la reutilización y la extensibilidad.

El problema de la herencia múltiple que ocurre en lenguajes como C++, estos problemas vienen debido a las situaciones en las que potencialmente se pueden producir conflictos entere métodos y variables con igual nombre, eventualmente con comportamientos diferentes, esto crea un desajuste cognitivo que va en contra de los principios de la programación orientada a objetos.

* **Propuestas de solución**:
  1. Primera solución:
     + Diseñar mediante el proceso de abstracción una clase que represente las personas y de ella se obtendrá una que represente al ciclista.
     + Diseñar un modelo de herencia para la bicicleta y sus diferentes tipos.
     + Cada entidad o elemento del sistema poseerá un método para proveer una salida de datos, correspondiente a su estado.
  2. Segunda solución:
     + Diseñar mediante el proceso de abstracción una clase que represente las personas y de ella se obtendrá una que represente al ciclista.
     + Diseñar un modelo de herencia para la bicicleta y sus diferentes tipos.
     + Se provee con la herencia de interfaz un comportamiento común, tanto para las entidades que se ejecutarán como para las que poseerán una salida de datos, siendo estos dos comportamientos interfaces diferentes.
* **Desestimaciones**

Se desestima la primera solución por no cumplir con las directrices de la programación orientada a objetos, ya que no es un código mantenible si a la hora de cambiar la salida de datos o el método en el que se muestra hay que cambiar cada una de las clases que van a mostrar los datos.

* **Solución escogida**

Se ha elegido la segunda solución porque se cree que se cumple con el paradigma de la programación orientada a objetos.

Con esta solución se ha podido realizar la práctica cumpliendo con las especificaciones de esta, y superando las pruebas realizadas para garantizar que se cumple con la especificación y que su funcionamiento es correcto (dentro del ámbito de las pruebas desarrolladas para esta práctica).

**2ºProblema**

**3ºProblema**

Al introducir una carretera por la que circula el ciclista y que afecta al estado de la misma, nos hemos encontrado con una serie de consideraciones a tener en cuenta. Como él quien se va a encargar de modificar el estado de la bicicleta. Para ello se nos ocurrieron dos soluciones.

* **Propuesta de solución**

1. Primera solución
   * + Crear un mapa de carreteras que tuviera como valor una clase tramo. Esa clase tendría el km inicial, la pendiente, la dirección del viento y la velocidad del viento en ese tramo.
     + Luego ese mapa se añadiría como un atributo a la clase Bicicleta y esta se ocuparía de modificar la velocidad en proporción del tramo de la carretera en el que estuviera circulando.
2. Segunda solución
   * + Crear un mapa de carreteras que tuviera como valor una clase tramo. Esa clase tendría el km inicial, la pendiente, la dirección del viento y la velocidad del viento en ese tramo.
     + Crear una nueva clase llamada FactoreExternos que heredaría de la interface ObjetosQueSeEjecutan. Esta clase tendría como atributos el mapa de carreteras y un objeto tipo Bicicleta.
     + Esta clase se ocuparía de mirar el tramo en el que se encuentra la bicicleta y modificar la velocidad en proporción al tramo en el que se encuentra.
     + En el manager se tendría que ejecutar la clase.

* **Desestimaciones**

Se desestimo la primera solución ya que añadía nuevos atributos a la clase bicicleta lo que la hacía menos portable y reutilizable. Además nos parecía que al ser unos factores que no tenían nada que ver con la bicicleta no tenía sentido que se incorporara a esta

* **Solución Escogida**

Se eligió la segunda solución, ya que nos pareció mas adecuado que al ser unos elementos externos, estos fueran externos a la bicicleta. Además así le añadía un grado más alto de portabilidad y reutibilidad.

1. **Conclusión**

Al realizar esta práctica hemos aprendido la parte de la programación orientada a objetos que compete la herencia, que es algo fundamental en este paradigma de programación. Hemos trabajado los diferentes tipos de herencia, simple, múltiple, y de comportamiento o interfaz.

Se ha profundizado en la vinculación dinámica y estática en Java, cuya repercusión en la memoria puede llegar a ser notable en problemas de software de gran envergadura.

* **Líneas Futuras**

De cara al futuro se espera seguir mejorando los conocimientos sobre el lenguaje java y la programación orientada a objetos, con el fin de construir software que cumpla con los principios del software y de calidad.

1. **Bibliografía**

**Estructura del documento actual:**

* Transparencias sobre la estructura de las prácticas, del profesor Francisco Javier Crespo.

**Programación orientada a objetos:**

* Piensa en Java 4ª Edición, Bruce Eckel
* <http://es.wikipedia.org/wiki/Clase_%28inform%C3%A1tica%29>
* <http://es.wikipedia.org/wiki/Metadato>
* <http://es.wikipedia.org/wiki/Objeto_%28programaci%C3%B3n%29>
* <http://es.wikipedia.org/wiki/Herencia_%28programaci%C3%B3n_orientada_a_objetos%29>

**Física:**

* <http://www.didactika.com/fisica/cinematica/movimiento_rectilineo_uniformemente_variado.html>