Programación II. Algoritmos y Estructuras de datos II. 2024

Programación II. Algoritmos y EDD II. 2024

Agenda 11 de Marzo 2024:

- Generalidades (forma de evaluación, encuentros)
- Objetivos del curso
- Temas centrales
- Lenguaje de Programación

Programación II. Algoritmos y EDD II. 2024

Generalidades

Forma de evaluación

- Examen parcial +
- Entrega de un TP con examen (ejercicios con investigación).

  Se podrá recuperar UNO de ambos.

  Recuperatorio / final optativo: cronograma. Final regular: cronograma

Programación II. Algoritmos y EDD II. 2024

Generalidades

Criterios de evaluación

En esta materia evaluaremos:

- La estrategia que se utiliza para resolver el problema: se espera que el alumno explique claramente los pasos que planea seguir para resolver el problema.
- La corrección del programa. El programa debería poder ser ejecutado y resolver el problema. Su estructura debe corresponder a su estrategia.
- La claridad y legibilidad de su programa. El programa debe ser fácilmente legible. Esto puede mejorarse con comentarios adecuados. La eficiencia de su programa. El programa no debe ser innecesariamente complicado.

Programación II. Algoritmos y EDD II. 2024

Generalidades

Criterios de evaluación

- Estar seguro de que se entiende el problema que debe resolver antes de empezar a trabajar.
- Establecer una clara estrategia de resolución. Debemos saber con qué datos contamos y para qué nos pueden servir. Nunca se debe empezar a codificar antes de tener un plan.
- Esto implica, entre otras cosas, definir las estructuras de datos que serán necesarias para resolver el problema.
- Tener presente: un programa va a ser leído no sólo por una computadora sino muy probablemente también por personas. Claridad y comentarios en el código.

Programación II. Algoritmos y EDD II. 2024

Generalidades

Encuentros

- Presenciales
- Eventualmente virtuales (con anuncio anticipado)
- Exámenes presenciales

Programación II. Algoritmos y EDD II. 2024

Objetivos del curso

El objetivo del curso de Programación II es introducir el concepto de tipos de datos abstractos (TDA), sus especificaciones, implementaciones y aplicaciones.

Se utilizará el lenguaje de programación Java como lenguaje de soporte, siendo el lenguaje un medio para aplicar los contenidos. Se trabajará con estructuras de datos conocidas de cursos previos, como son las pilas y colas entre otras y se continuará con otras estructuras más complejas como son los árboles y grafos.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

- **Algoritmia**
- **Tipos de datos abstractos**
  - Conceptos
  - Especificaciones
  - Implementación de los tipos de datos abstractos
    - Estructuras estáticas
    - Estructuras dinámicas
  - Arboles
  - Grafos
- **Análisis de eficiencia de algoritmos. Complejidad temporal y espacial.**

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia

- **Que es un algoritmo ?**
  - Algoritmo / Programa
  - Entrada / Salida
  - Problema algorítmico / Solución algorítmica
- **Que es un programa?**

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia: Análisis, Diseño e Implementación de Algoritmos

Que es un Algoritmo?

Procedimiento: secuencia de pasos elementales, precisos, que pueden ejecutarse para resolver una tarea

Algoritmo: Procedimiento + finitud

Procedimiento efectivo

- **¿Un programa es un procedimiento?**
- **¿Un programa es un procedimiento efectivo?**

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia: Análisis, Diseño e Implementación de Algoritmos

Algoritmo: Procedimiento + finitud: Procedimiento efectivo

Programa: Procedimiento

Los programas son una forma computacional de expresar algoritmos (más precisamente colecciones de algoritmos).

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia: Análisis, Diseño e Implementación de Algoritmos

Un problema algorítmico consiste de:

una caracterización de las entradas válidas, colección de los potenciales conjuntos de entrada

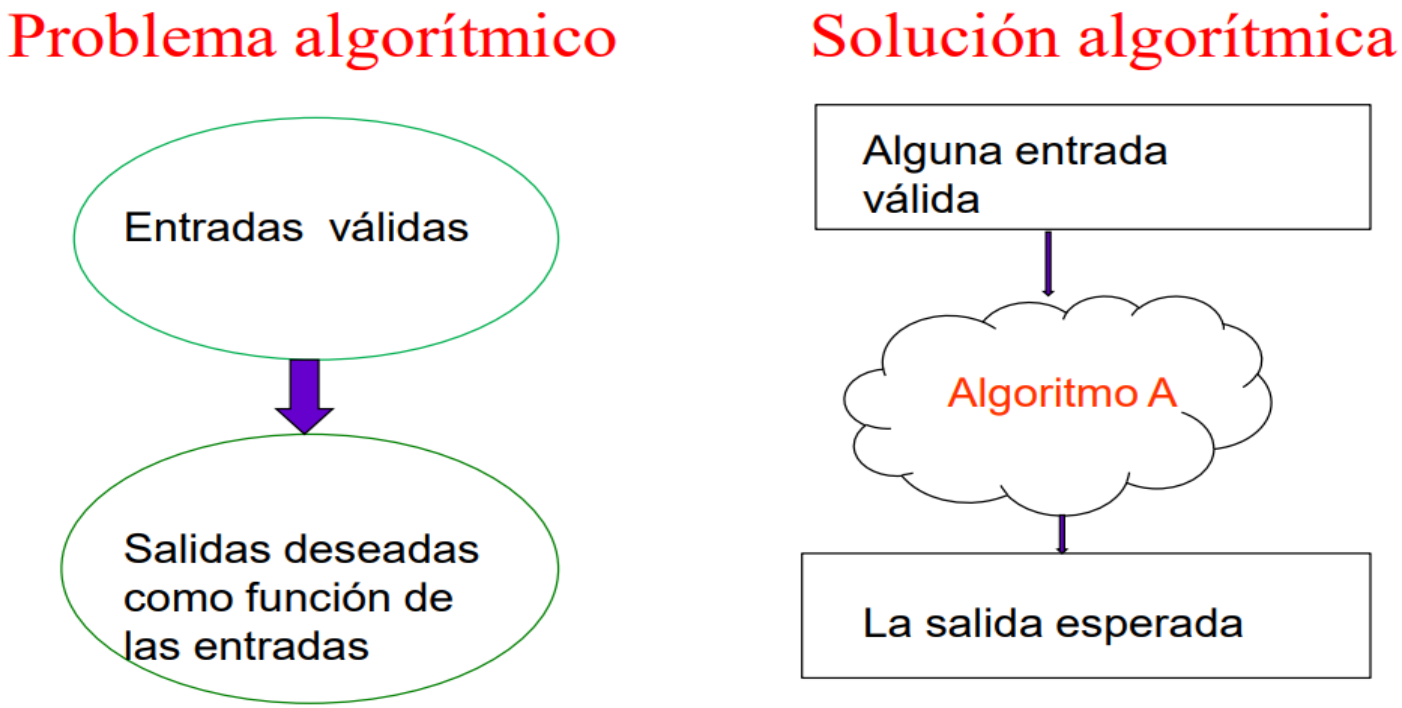
y

una especificación de las salidas deseadas como función de las entradas

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia: Análisis, Diseño e Implementación de Algoritmos

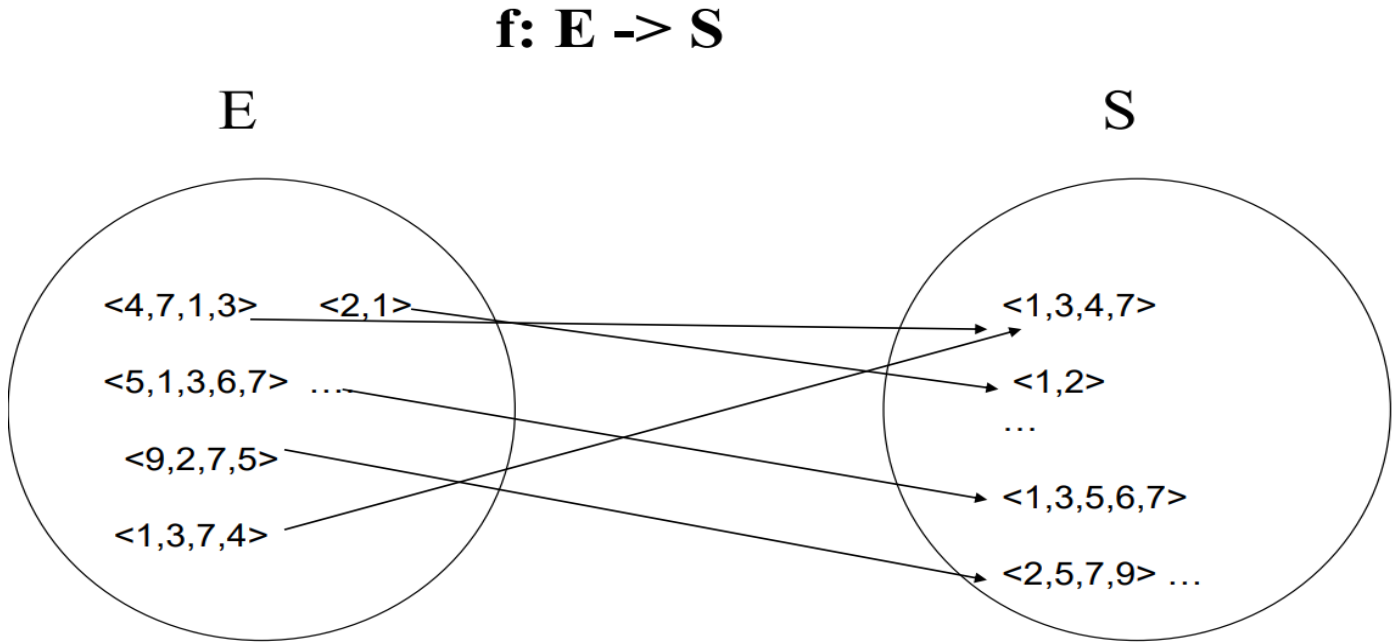


Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia: Análisis, Diseño e Implementación de Algoritmos

Ejemplo: algoritmo de Ordenamiento



Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia: Análisis, Diseño e Implementación de Algoritmos

De que hablamos cuando hablamos de los algoritmos relacionados a la Inteligencia Artificial?

La IA y la algoritmia están estrechamente relacionadas, ya que la IA utiliza algoritmos como herramientas fundamentales para realizar diversas tareas inteligentes.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia: Análisis, Diseño e Implementación de Algoritmos

La Algoritmia y la Inteligencia Artificial

Una de las partes muy preponderante en el desarrollo de las “tareas inteligentes” es el **entrenamiento del algoritmo**: en esta etapa se alimentan los datos preprocesados al algoritmo, se ajustan sus parámetros internos (con técnicas de optimización para minimizar una función de pérdida o maximizar una función), dependiendo del tipo de problema que se esté abordando.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Algoritmia: Análisis, Diseño e Implementación de Algoritmos

En resumen:

- **Concepto de Algoritmo**
- **Diferencia Algoritmo / Programa**
- **Algoritmos y la IA**

Volveremos con Algoritmos cuando analicemos costos y complejidad temporal

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- Conceptos
- Especificaciones
- Implementación de los tipos de datos abstractos
  - Estructuras estáticas
  - Estructuras dinámicas
- Arboles
- Grafos

Lenguaje de Programación

Programación II. Algoritmos y EDD II. 2024

Temas centrales

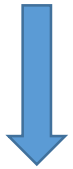
Tipos de datos abstractos

- **Conceptos**

Programación II. Algoritmos y EDD II. 2024

Tipos de datos abstractos. Conceptos

Tipo de dato en lenguajes de programación



Tipo de dato = { Dominio, Operaciones}

Programación II. Algoritmos y EDD II. 2024

Tipos de datos abstractos. Introducción

Tipos primitivos en un lenguaje de programación

short int

Dominio: -32768 .. 32768

Operaciones: +, - ,* , /,--, ++, > , < , >=, <=, ==, != , %,…

Int

Dominio: -2147483647..2147483647

Operaciones: +, - ,* , /,--, ++, > , < , >=, <=, ==, != , %,…

bool

Dominio: {true, false}

Operaciones: and (&&), or (||),…

Programación II. Algoritmos y EDD II. 2024

Tipos de datos abstractos. Introducción

Tipos primitivos en un lenguaje de programación

Características de los tipos primitivos:

- Ocultamiento de información, no sabemos cómo está implementado el tipo
- Conjunto preestablecido de operaciones, que usamos sin saber cómo están implementadas

Programación II. Algoritmos y EDD II. 2024

Tipos de datos abstractos. Introducción

Tipos primitivos en un lenguaje de programación

Tipos de datos y evolución en los lenguajes de programación

Tipos de datos primitivos + mecanismos de estructuración de tipos de datos

- para agrupar componentes homogéneas (arreglos)
- para agrupar componentes heterogéneas (registros)

Programación II. Algoritmos y EDD II. 2024

Tipos de datos abstractos. Introducción

Tipos primitivos en un lenguaje de programación

Tipos de datos y evolución en los lenguajes de programación

Tipos de datos primitivos + mecanismos de estructuración de tipos de datos

No garantizan:

- ocultamiento de información
- operar solo a través de un conjunto de operaciones preestablecidas

Programación II. Algoritmos y EDD II. 2024

Tipos de datos abstractos. Introducción

Tipos de datos y evolución en los lenguajes de programación

Tipos de datos abstractos

Tipos construidos por el usuario (programador) con los mecanismos provistos por el lenguaje de programación con las características de los tipos primitivos

DOMINIO + OPERACIONES

UNIDAD SINTÁCTICA ENCAPSULAMIENTO

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- Definición
- Especificaciones
    - Formales e informales
- Abstracciones

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Definición

Conjunto de valores sobre los que se aplica un conjunto dado de operaciones que cumplen determinadas propiedades.

Que significa **abstracto**? En este contexto, el término corresponde al hecho de que los valores de un tipo pueden ser manipulados mediante sus operaciones.

Que debemos conocer de las operaciones? Las propiedades que estas cumplen, sin necesidad de conocer su implementación

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Definición

En el caso del tipo primitivo Int de los lenguajes de programación, debemos saber:

- Cualquier programa puede efectuar x+y, siendo x e y variables de tipo entero con la certeza que calculará siempre la suma de los enteros x e y independientemente de su representación interna.
- La manipulación de los objetos de un tipo sólo depende del comportamiento descripto en su **especificación** y es independiente de su **implementación**.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Especificaciones

La especificación de un TDA consiste en establecer propiedades que lo definen. La misma debe ser:

- Precisa: debe decir lo imprescindible
- General: adaptable a diferentes conceptos
- Legible: que sirva como instrumento de comunicación entre:
  - el especificador y los usuarios del tipo
  - el especificador y el implementador del tipo
- No ambigua, evitando posteriores problemas de interpretación

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Especificaciones

La especificación de un TDA que es única, define el comportamiento a cualquier usuario que lo necesite, depende del grado de formalismo de la misma.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Especificaciones

Para describir el comportamiento de un TDA hay diversas maneras:

- Especificaciones informales
- Especificaciones formales

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Especificaciones

Para describir el comportamiento de un TDA hay diversas maneras:

- Especificaciones informales:
  - Lenguaje natural, falta de precisión
  - Diagramas, casos, tablas, en sus diferentes versiones

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Especificaciones

Para describir el comportamiento de un TDA hay diversas maneras:

- Especificaciones informales. **Ventajas**:
  - Facilidad de comprensión (asociado al lenguaje natural)
  - Flexibilidad (pueden adaptarse a diferentes contextos)
  - Rapidez en la escritura
  - En un ambiente informal de trabajo, rapidez para el trabajo en conjunto

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Especificaciones

Para describir el comportamiento de un TDA hay diversas maneras:

- Especificaciones informales. **Desventajas**:
  - Ambigüedad, puede dar espacio a diferentes interpretaciones
  - Falta de precisión, importante a la hora de implementarlo
  - Difícil de verificar y validar (en especial cuando se tienen pruebas automáticas)

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- Especificaciones formales
  - Permiten expresar sin ninguna ambigüedad las propiedades que cumplen las operaciones de un tipo, mediante expresiones rigurosas.
  - Existen distintos formalismos: las especificaciones algebraicas, lógica de predicados, etc

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- Especificaciones formales. Ventajas
  - **Precisión, Claridad**: ayuda a evitar ambigüedades y errores en la comprensión del tipo de dato abstracto. Esto permite una interpretación unívoca del comportamiento y las propiedades del tipo de dato.
  - **Verificación rigurosa**: facilita la detección temprana de errores y garantiza un comportamiento correcto del tipo de dato en diferentes contextos y situaciones.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- Especificaciones formales. Ventajas
  - **Reutilización y Modularidad**: facilita la reutilización y la modularidad del tipo de dato abstracto en diferentes partes de un sistema de software. Se facilita el mantenimiento y la evolución del sistema con el tiempo.
  - **Facilita la Implementación y la Prueba**: la especificación formal proporciona una guía clara y precisa para la implementación del tipo de dato abstracto, facilitando el desarrollo de código correcto y eficiente ayudando en la definición de casos de prueba.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- Especificaciones formales. Desventajas
  - Implican desafíos y costos adicionales que deben considerarse cuidadosamente en el proceso de desarrollo de software debido a que es un proceso complejo y laborioso, ya que requiere un alto nivel de rigor, formalidad y tiempo.
  - Las especificaciones formales tienden a ser más rígidas y menos flexibles que las especificaciones informales. Esto puede dificultar la adaptación a cambios en los requisitos o en el diseño del sistema.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- Especificaciones formales. Desventajas
  - Limitaciones de Herramientas y Soporte: aunque existen herramientas y técnicas para facilitar la especificación formal, pueden no estar tan ampliamente disponibles o maduras como las herramientas para el desarrollo de software convencional.
  - Costo de Mantenimiento: Mantener una especificación formal actualizada y alineada con los cambios en el sistema puede ser costoso y requerir un esfuerzo considerable, especialmente en sistemas grandes y complejos.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- **Abstracciones**

Volviendo sobre la característica de los tipos de datos “abstractos”, veamos que está significando la abstracción en nuestro contexto:

- Los niveles de abstracción dependen del nivel de detalle nos interese para la definición de un objeto.
- En un **nivel alto**, nos interesa manejar un electrodoméstico inteligente, un auto, sin necesidad de conocer los detalles del funcionamiento de la inteligencia del electrodoméstico o del motor del auto. En un **nivel bajo** (el del fabricante o del ingeniero mecánico) es necesario conocer perfectamente los detalles.
- Así, los detalles ocultos de un objeto deben estar implementados. El nivel de abstracción del implementador del objeto es por lo tanto más bajo que el nivel de abstracción del usuario.

Trabajaremos con todos estos niveles de abstracción durante este curso.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos

- **Abstracciones**

Trabajaremos con todos estos niveles de abstracción durante este curso, que significa?

- En principio vemos el comportamiento del tipo de dato abstracto para definir su funcionalidad
- Luego, implementaremos la funcionalidad

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Resumen

En la definición de un TDA, se separa el qué hace un TDA del cómo lo hace.

Para eso:

- se pueden definir TDA y utilizarlos sin necesidad de conocer cómo se hace cada una de las operaciones. Para una misma definición de TDA puede haber diferentes maneras de realizar lo definido por ese TDA.
- A la definición del TDA la llamaremos **especificación** y a la forma de llevar a cabo lo definido por un TDA lo denominaremos **implementación**.
- Por lo tanto, para un mismo TDA vamos a tener diferentes implementaciones.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal

Un ejemplo de una especificación de un **TDA Pila**:

El primer TDA a especificar será el tipo de dato abstracto que describe el comportamiento de una estructura **pila**.

La pila es una estructura que permite almacenar conjuntos de valores, eliminarlos y recuperarlos, con la particularidad de que el elemento que se recupera o elimina es el último que ingresó.

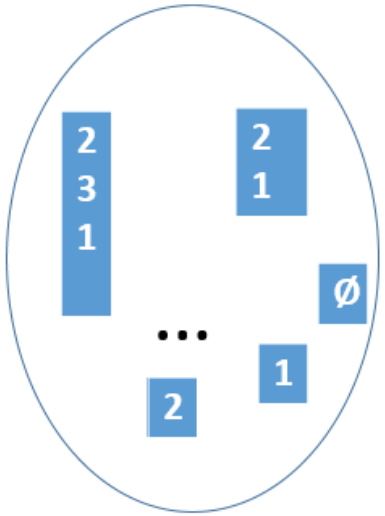
Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal

TDA Pila

Ejemplo de un Dominio para Pila



Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal

Un ejemplo de una especificación de un TDA Pila. Conociendo el Dominio para el tipo Pila

- Cómo construyo el conjunto de operaciones?
- Cual es el conjunto mínimo de operaciones para definir al TDA?
    - Constructoras básicas
    - Observadoras
    - Modificadoras

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal

Un ejemplo de una especificación de un TDA Pila.

- Cual es el conjunto mínimo de operaciones para definir al TDA?
  - **Constructoras básicas**: son aquellas operaciones SUFICIENTES que permiten generar todos los valores del tipo, desde la pila vacia a la pila de n elementos
    - **Apilar**: permite agregar un elemento a la pila. Se supone que la pila está inicializada.
    - **InicializarPila**: permite inicializar la estructura de la pila.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal

Un ejemplo de una especificación de un TDA Pila.

- Cual es el conjunto mínimo de operaciones para definir al TDA?
    - **Observadoras**: son aquellas operaciones que obtienen información del objeto sin modificarlo
        - **Tope:** permite conocer cuál es el último elemento ingresado a la pila. Se supone que la pila no está vacía.
        - **PilaVacía:** indica si la pila contiene elementos o no. Se supone que la pila está inicializada.

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal

Un ejemplo de una especificación de un TDA Pila.

- Cual es el conjunto mínimo de operaciones para definir al TDA?
    - **Modificadoras**: son aquellas operaciones que modifican objeto, devolviendo otro objeto
        - **Desapilar**: permite eliminar el último elemento agregado a la pila. Se supone como precondición que la pila no esté vacía.

Programación II. Algoritmos y EDD II. 2024

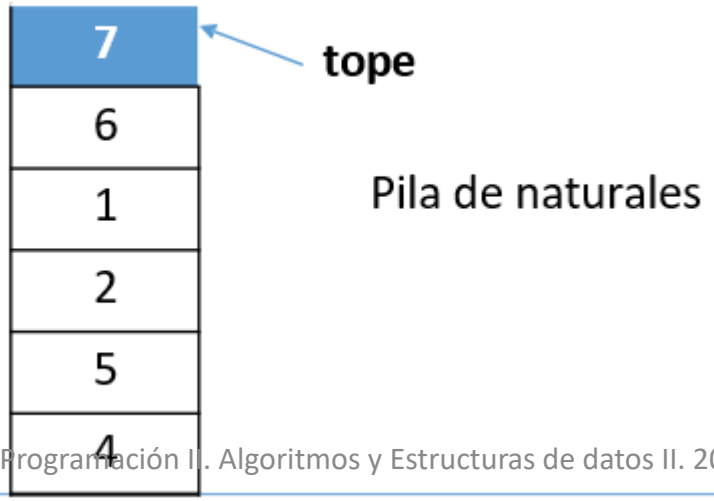
Temas centrales

Tipos de datos abstractos. Una especificación Informal

TDA **Pila**

Secuencia de elementos del mismo tipo en la que las operaciones se realizan en una posición distinguida llamada tope.

LIFO (last-in first-out)

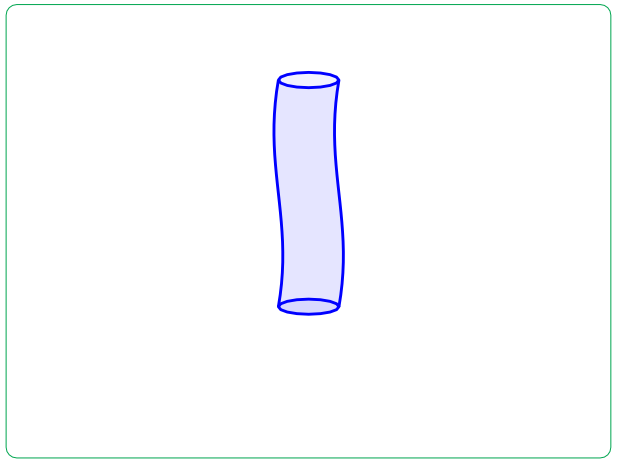


Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal

TDA Pila



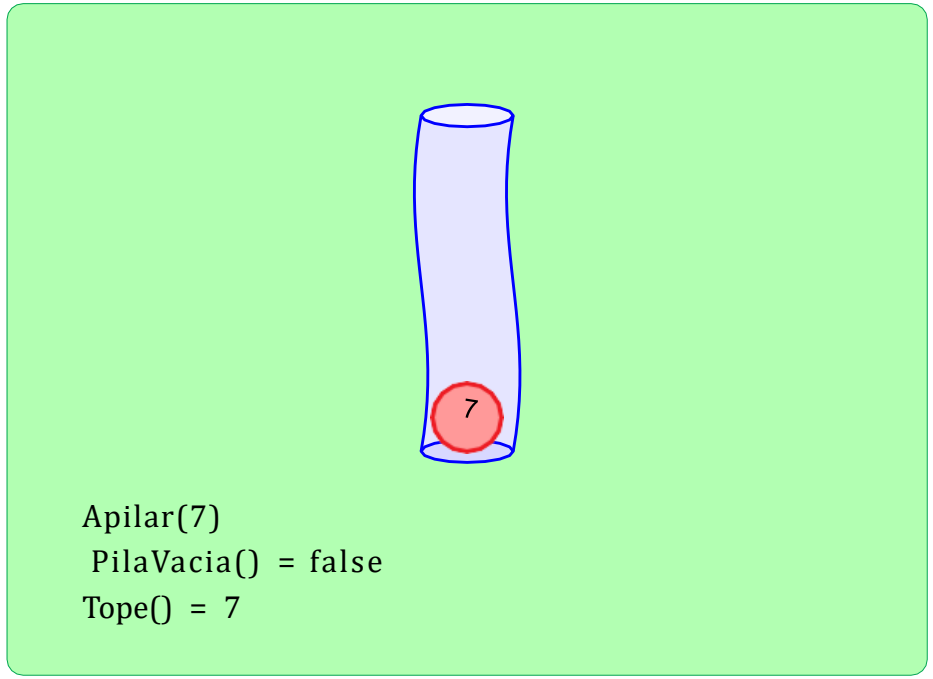
$$\text{PilaVacia}() = \text{true}$$

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal

TDA Pila



Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal. TDA Pila

Para definir el comportamiento de la pila vamos a utilizar la siguiente lista de operaciones:

- **InicializarPila: permite inicializar la estructura de la pila.**
- **Apilar: permite agregar un elemento a la pila. Se supone que la pila está inicializada.**
- **Desapilar: permite eliminar el último elemento agregado a la pila. Se supone como precondición que la pila no esté vacía.**
- **Tope: permite conocer cuál es el último elemento ingresado a la pila. Se supone que la pila no está vacía.**
- **PilaVacía: indica si la pila contiene elementos o no. Se supone que la pila está inicializada.**

Programación II. Algoritmos y EDD II. 2024

Temas centrales

Tipos de datos abstractos. Una especificación Informal. TDA Pila

Para definir el comportamiento de la pila vamos a utilizar la siguiente lista de operaciones:

- **InicializarPila: permite inicializar la estructura de la pila*. Constructora básica***
- **Apilar: permite agregar un elemento a la pila. Se supone que la pila está inicializada. *Constructora básica***
- **Desapilar: permite eliminar el último elemento agregado a la pila. Se supone como precondición que la pila no esté vacía. *Modificadora***
- **Tope: permite conocer cuál es el último elemento ingresado a la pila. Se supone que la pila no está vacía. *Observadora***
- **PilaVacía: indica si la pila contiene elementos o no. Se supone que la pila está inicializada. *Observadora***

Programación II. Algoritmos y EDD II. 2024

Temas centrales. 2da parte clase 1

Tipos de datos abstractos

- **Implementación de los tipos de datos abstractos**

El lenguaje de programación JAVA

Programación II. Algoritmos y EDD II. 2024

Temas centrales

JAVA

Programación II. Algoritmos y EDD II. 2024

- Introducción al lenguaje Java
  - Tipos de datos
  - Variables
  - Estructuras de control
  - Estructuras de datos
- Métodos y clases
  - Pasajes por valor y por referencia

Programación II. Algoritmos y EDD II. 2024

El lenguaje Java comenzó a desarrollarse a comienzos de los años noventa del siglo pasado.

Originalmente se tituló *Oak* por un roble que estaba situado fuera de la oficina de uno de sus creadores.

Luego fue rebautizado *Green* y finalmente *Java*, por el café de Indonesia.

Programación II. Algoritmos y EDD II. 2024

- El lenguaje Java comenzó a desarrollarse a comienzos de los años noventa del siglo pasado. Originalmente se tituló *Oak* por un roble que estaba situado fuera de la oficina de uno de sus creadores.
- Luego, fue rebautizado *Green* y finalmente *Java*, por el café de Indonesia.

  Java es un lenguaje orientado a objetos con una sintaxis similar a la de *C / C++*. Es decir, en Java los objetos son los ciudadanos de primera categoría.

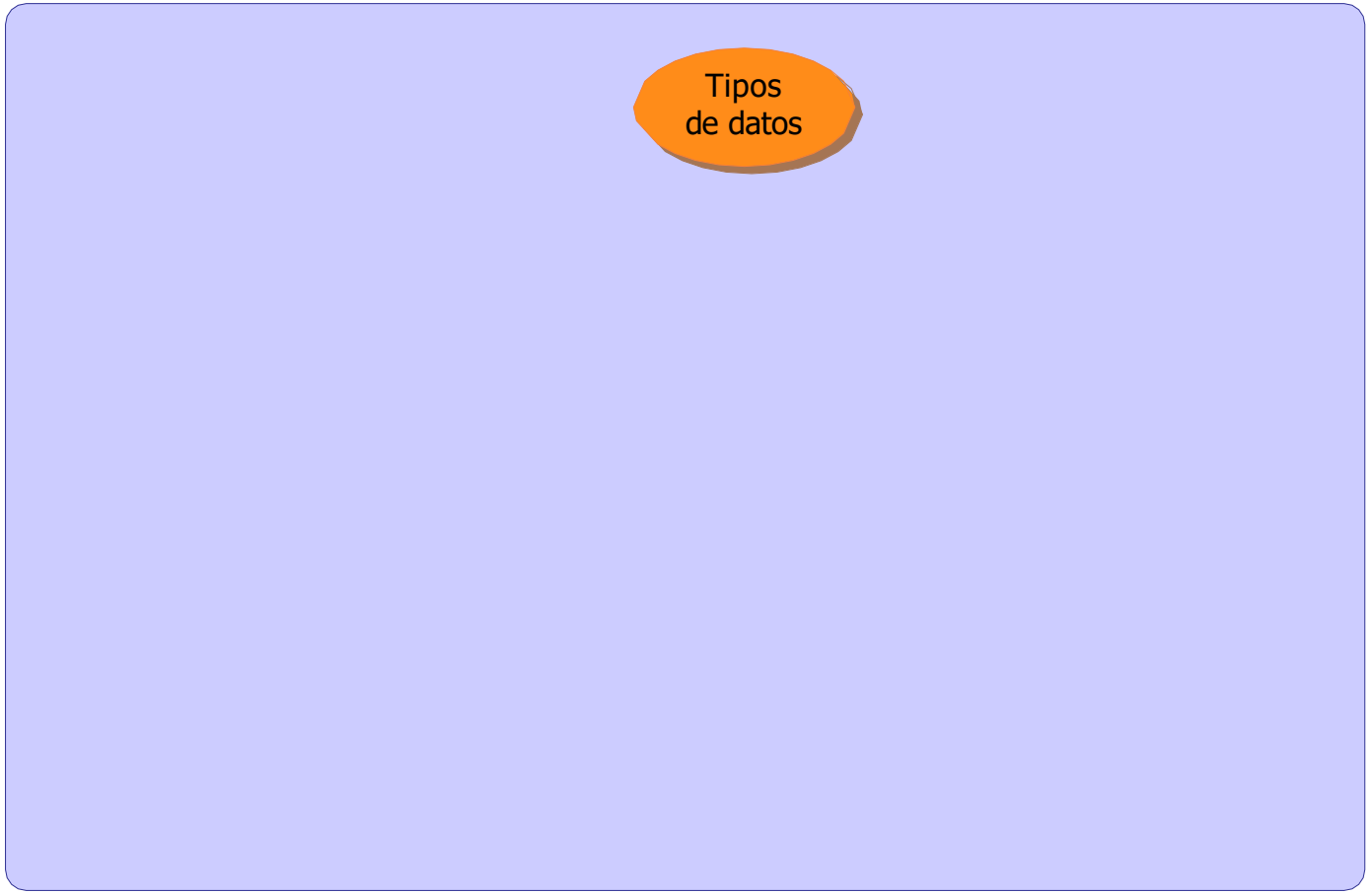
Programación II. Algoritmos y EDD II. 2024

- El lenguaje Java comenzó a desarrollarse a comienzos de los años noventa del siglo pasado. Originalmente se tituló *Oak* por un roble que estaba situado fuera de la oficina de uno de sus creadores. Luego, fue rebautizado *Green* y finalmente *Java*, por el café de Indonesia.
- Java es un lenguaje orientado a objetos con una sintaxis similar a la de *C / C++*. Es decir, en Java los objetos son los ciudadanos de primera categoría.
- Los programas Java son típicamente más rápidos que sus equivalentes Python, aunque el desarrollo es más lento, debido a algunas características de Python como la incorporación de algunos tipos de datos de alto nivel y el tipado dinámico.

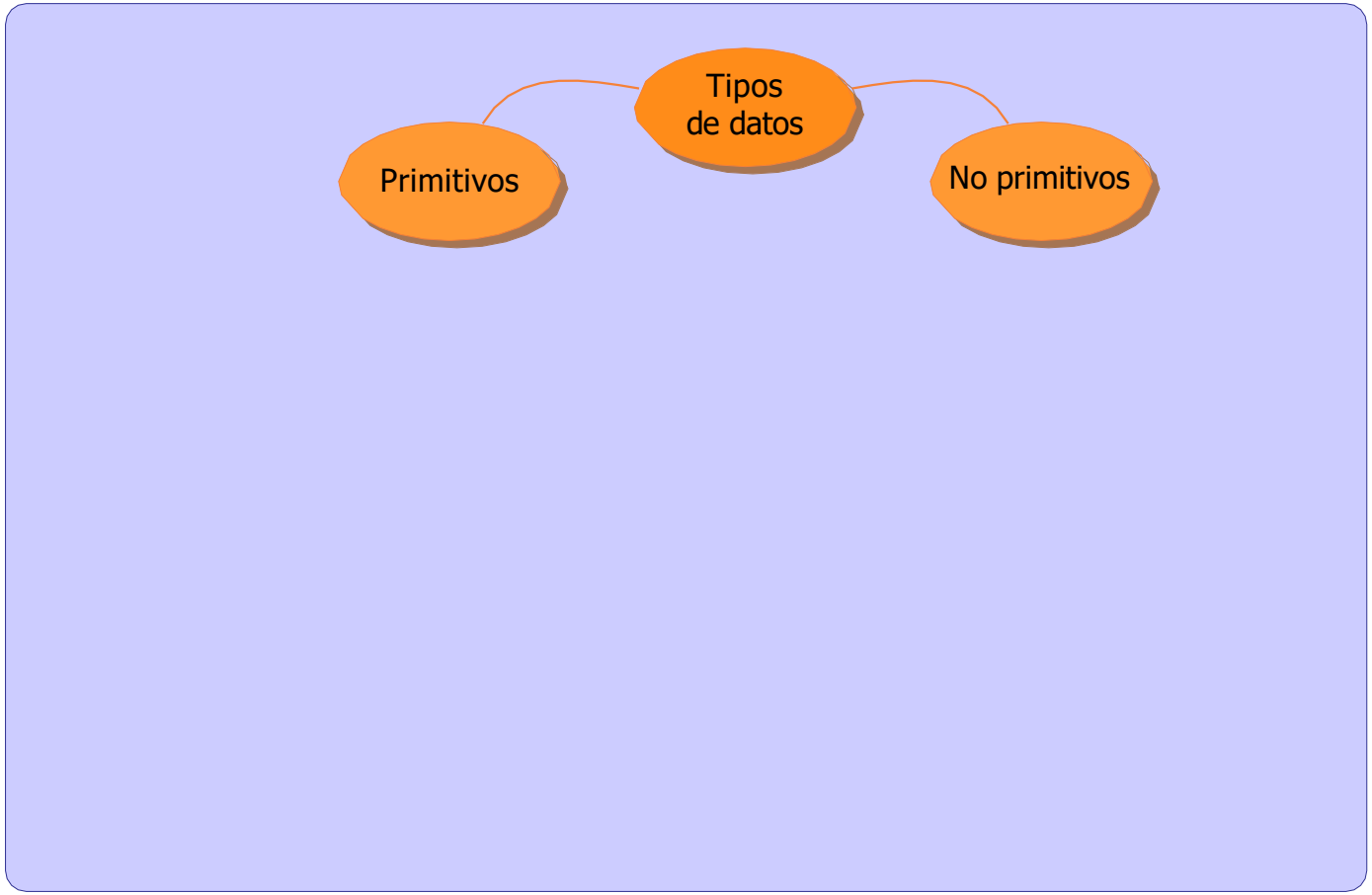
Programación II. Algoritmos y EDD II. 2024

1. Introducción al lenguaje Java
   - Tipos de datos
   - Variables
   - Estructuras de control
   - Estructuras de datos
2. Métodos y clases
   - Pasajes por valor y por referencia

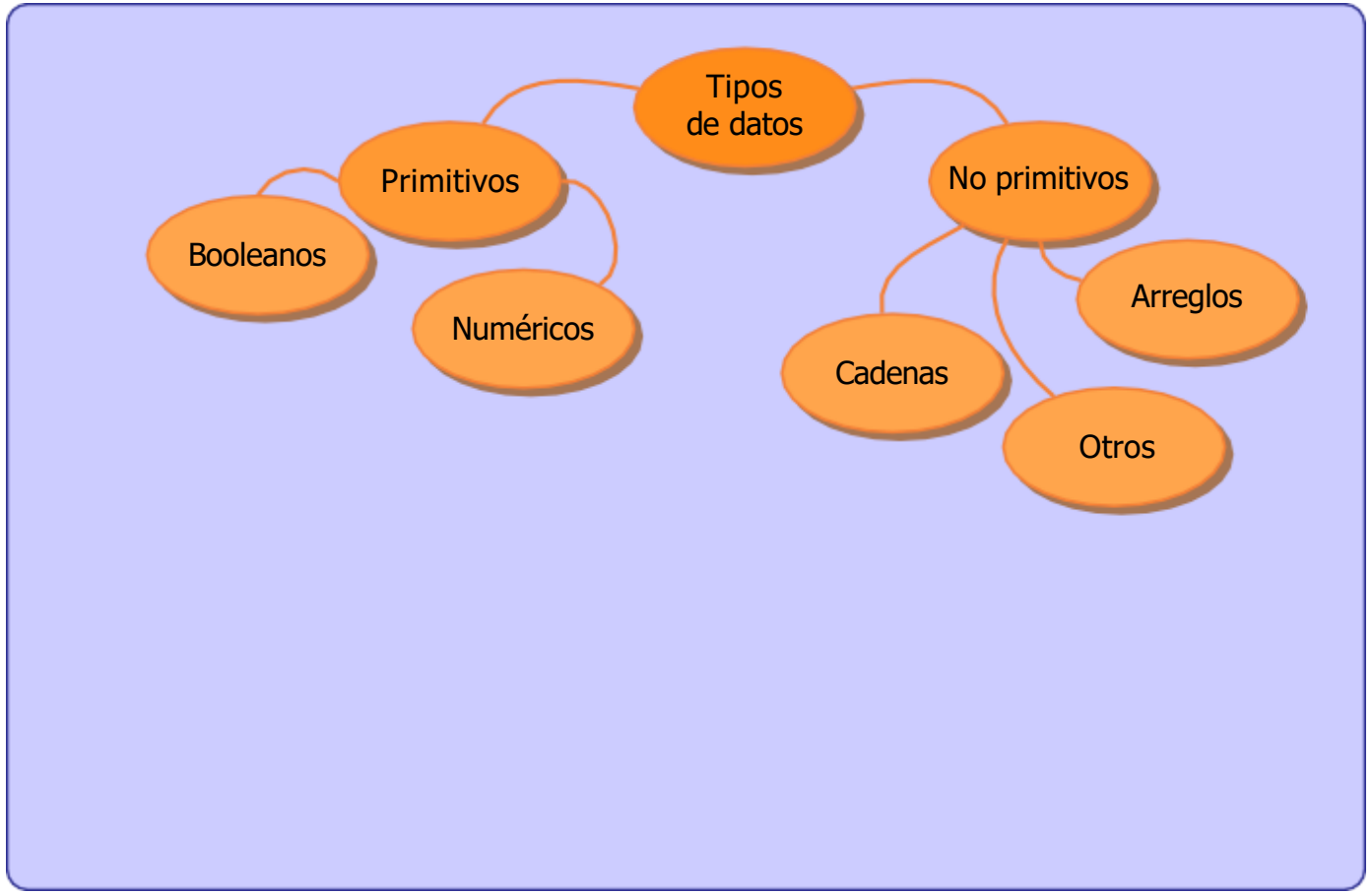
Programación II. Algoritmos y EDD II. 2024



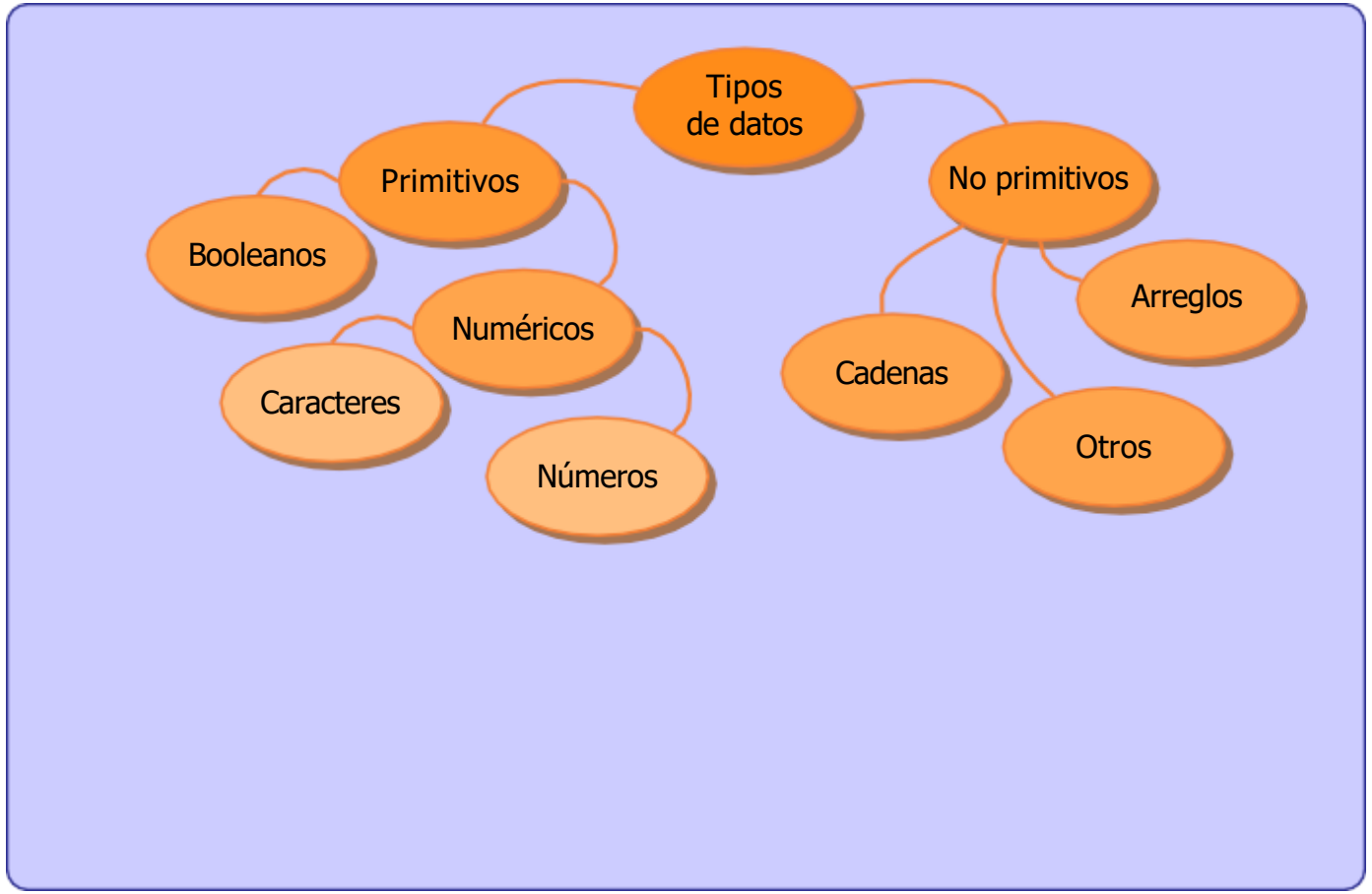
Programación II. Algoritmos y EDD II. 2024



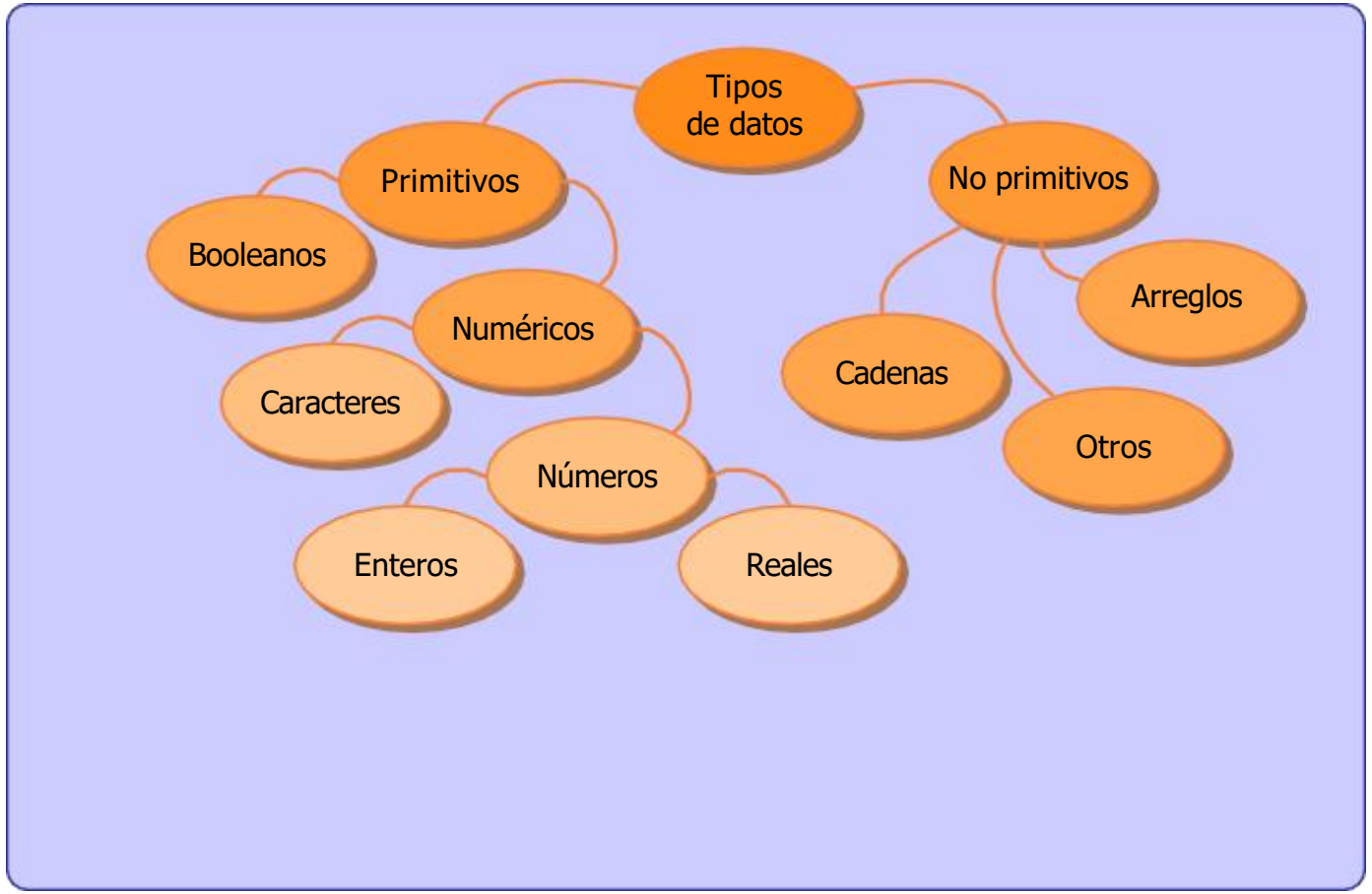
Programación II. Algoritmos y EDD II. 2024



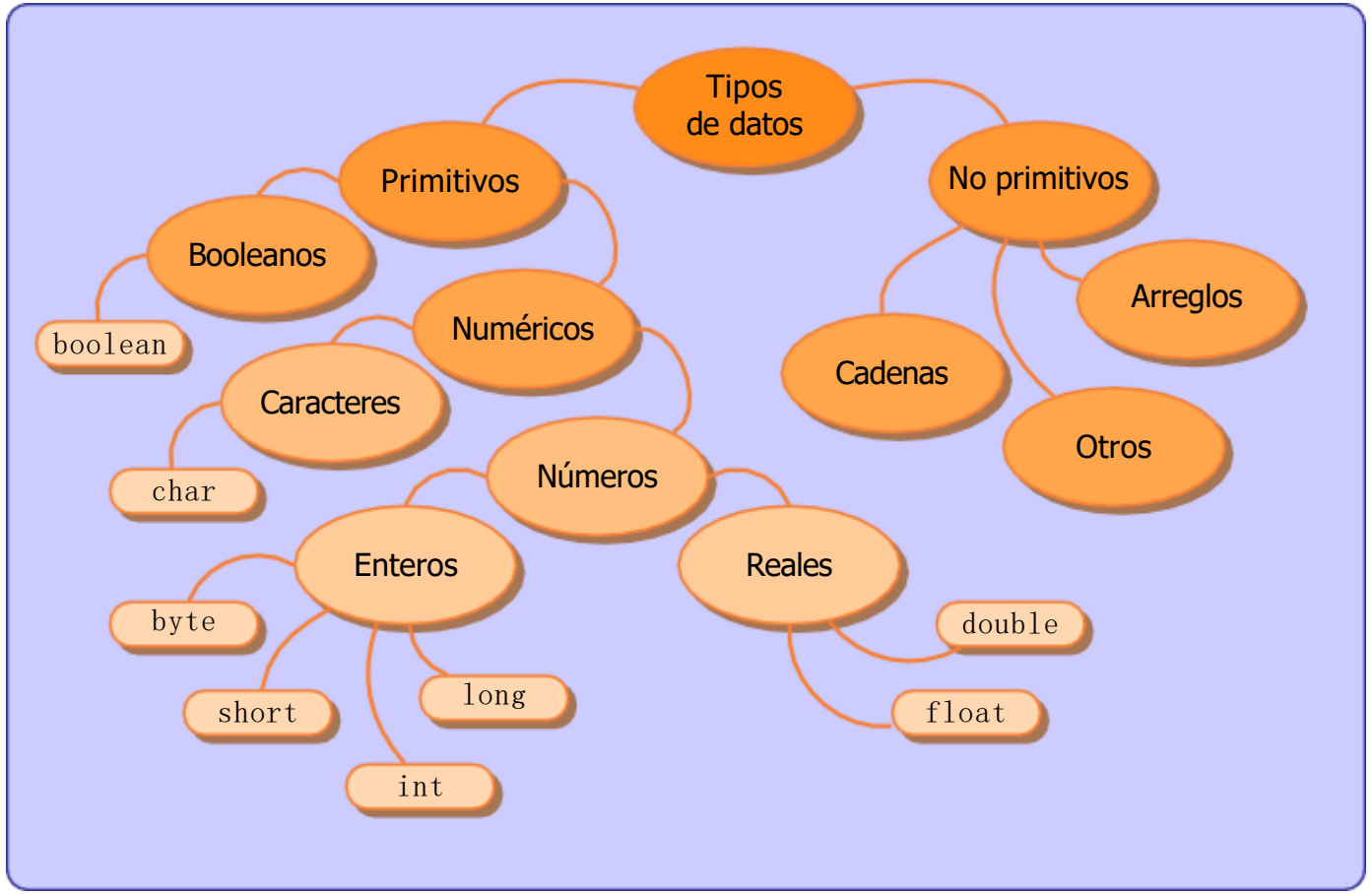
Programación II. Algoritmos y EDD II. 2024



Programación II. Algoritmos y EDD II. 2024



Programación II. Algoritmos y EDD II. 2024



Programación II. Algoritmos y EDD II. 2024

1. Introducción al lenguaje Java
   Tipos de datos
   - Variables
   - Estructuras de control
   - Estructuras de datos

2. Métodos y clases
   - Pasajes por valor y por referencia

Programación II. Algoritmos y EDD II. 2024

- Java es un lenguaje *estáticamente tipado*. Esto significa que el tipo de cada variable es conocido en el momento de la compilación.
- Debido a esto, cada variable debe ser declarada con su tipo de datos correspondiente antes de ser utilizada. No se puede cambiar el tipo de datos de una variable durante la ejecución de un programa.
- Recordemos que una variable es un área de memoria reservada. En otras palabras, el nombre de la variable es el nombre de la asignación de memoria.
- Como vimos, tenemos tipos de datos primitivos y tipos de datos no primitivos. Por ahora discutiremos los tipos de datos primitivos.
- Adoptaremos la siguiente convención: utilizaremos un recuadro verde para la sintaxis abstracta y un recuadro amarillo para los ejemplos.

```
Este formato se utilizará para la sintaxis abstracta
```

```
Este formato se utilizará para los ejemplos concretos
```

Definición de variables

Definición de variables

- El esquema general para una variable es

```
<tipo> <nombre de variable>;
```

Por ejemplo:

```
int contador;
```

El esquema para varias variables del mismo tipo es

```
<tipo> <nombre de variable>,…,<nombre de variable>;
```

Por ejemplo:

```
int contador, cantidad;
```

Definición de variables

- El esquema general para una variable es

```
<tipo> <nombre de variable>;
```

Por ejemplo:

```
int contador;
```

El esquema para varias variables del mismo tipo es

```
<tipo> <nombre de variable>,…,<nombre de variable>;
```

Por ejemplo:

```
int contador, cantidad;
```

- Puede declararse una variable y asignársele un valor al mismo tiempo:

```
int contador = 0;
char c = 'f';
```

Operadores

Operadores

- Operadores aritméticos
  - `+` suma num × num → num.
  - `-` resta num × num → num.
  - `/`, `÷` división y resto num × num → num.
  - `*` operador de multiplicación num × num → num.
  - `++`, `--` incremento / decremento unitario num → num.

Operadores

- Operadores aritméticos
  - `+` suma num × num → num.
  - `-` resta num × num → num.
  - `/`, `÷` división y resto num × num → num.
  - `*` operador de multiplicación num × num → num.
  - `++`, `--` incremento / decremento unitario num → num.
- Op
  - `>`, `<`, `>=`,`<=`, `==`, `!=` comparadores val × val → bool.

Operadores

- Operadores aritméticos
  - `+` suma num × num → num.
  - `-` resta num × num → num.
  - `/`, `÷` división y resto num × num → num.
  - `*` operador de multiplicación num × num → num.
  - `++`, `--` incremento / decremento unitario num → num.
- Op
  - `>`, `<`, `>=`,`<=`, `==`, `!=` comparadores val × val → bool.
- Op
  - `!` negación bool → bool.
  - `||`, `&&` disyunción y conjunción bool × bool → bool.

Estructuras de control

Estructuras de control

- Bloques: los bloques de sentencias se delimitan con llaves ({ y }). Se comienza un bloque con una llave de apertura (izquierda) y se finaliza con una llave de cierre (derecha.) Si el bloque consta de una única sentencia, pueden omitirse las llaves. Se suele indentar para facilitar la lectura. El compilador ignora las indentaciones.

Estructuras de control

- Bloques: los bloques de sentencias se delimitan con llaves ({ y }). Se comienza un bloque con una llave de apertura (izquierda) y se finaliza con una llave de cierre (derecha.) Si el bloque consta de una única sentencia, pueden omitirse las llaves. Se suele indentar para facilitar la lectura. El compilador ignora las indentaciones.
- Condicional simple. Tenemos dos posibilidades:

```
if <condición booleana> {
    bloque 1
}
```

```
if <condición booleana> {
    bloque 1
} else {
    bloque 2
}
```

Estructuras de control

Estructuras de control

- Condicional múltiple. La estructura es la siguiente:

```
switch <expresión entera> {
   case valor 1:
        sentencia 1;
        sentencia 2;
        …… break;
   case valor 2:
        sentencia 3;
        sentencia 4;
        …… break;
   case valor 3:
        sentencia 5;
        sentencia 6;
        …… break;
   default:
        sentencia 7;
        sentencia 8;
        ……
}
```

Comentario subjetivo: if-else y switch

Considero preferible la construcción if…else if…else:

```
switch (nota) {
case 10:
    System.out.printl("Sobresaliente");
    break;
case 9:
    System.out.printl("Distinguido");
    break;
case 8:
    System.out.printl("Muy bueno");
    break;
case 7:
    System.out.printl("Muy bueno");
    break;
case 6:
    System.out.printl("Bueno");
    break;
case 5:
    System.out.printl("Bueno");
    break;
case 4:
    System.out.printl("Suficiente");
default:
    System.out.printl("No alcanzó");
    break;
}
```

```
if(nota == 10)
    System.out.printl("Sobresaliente");
else if(nota == 9)
    System.out.printl("Distinguido");
else if(nota >= 7)
    System.out.printl("Muy bueno");
else if(nota >= 5)
    System.out.printl("Bueno");
else if(nota == 4)
    System.out.printl("Suficiente");
else
    System.out.printl("No alcanzó");
```

Estructuras de control

Estructuras de control

- Ciclos. Tenemos tres posibilidades:

```
while <condición booleana> {
        sentencia 1;
        sentencia 2;
        .....
}
```

```
do {
        sentencia 1;
        sentencia 2;
        .....
} while <condición booleana>
```

```
for <inicialización variable>; <condión booleana>; <variación variable> {
        sentencia 1;
        sentencia 2;
        .....
}
```

Estructuras de datos. Arreglos

Estructuras de datos. Arreglos

- Un arreglo se define con la siguiente sintaxis:

```
<tipo de datos del arreglo> [ ] <nombre del arreglo>;
```

Por ejemplo, la siguiente sentencia define un arreglo de enteros llamado vector:

```
int [ ] vector;
```

Estructuras de datos. Arreglos

- Un arreglo se define con la siguiente sintaxis:

```
<tipo de datos del arreglo> [ ] <nombre del arreglo>;
```

Por ejemplo, la siguiente sentencia define un arreglo de enteros llamado vector:

```
int [ ] vector;
```

- Es necesario especificar el tamaño del arreglo. Por ejemplo, si nuestro arreglo vector va a tener 20 posiciones, debemos escribir antes de comenzar a utilizarlo:

```
vector = new int [20];
```

Estructuras de datos. Arreglos

- Un arreglo se define con la siguiente sintaxis:

```
<tipo de datos del arreglo> [ ] <nombre del arreglo>;
```

Por ejemplo, la siguiente sentencia define un arreglo de enteros llamado vector:

```
int [ ] vector;
```

- Es necesario especificar el tamaño del arreglo. Por ejemplo, si nuestro arreglo vector va a tener 20 posiciones, debemos escribir antes de comenzar a utilizarlo:

```
vector = new int [20];
```

- También se puede hacer lo siguiente:

```
int [ ] valores = {1,2,3,4};
```

Aquí tendremos el vector [1,2,3,4].

Métodos

Métodos

- Los *métodos* pueden compararse con los procedimientos y funciones de los lenguajes estructurados. Un método tiene un tipo (que es el tipo del valor que devuelve; si no devuelve ningún valor, el método se declara como de tipo `void`.

Métodos

- Los *métodos* pueden compararse con los procedimientos y funciones de los lenguajes estructurados. Un método tiene un tipo (que es el tipo del valor que devuelve; si no devuelve ningún valor, el método se declara como de tipo `void`.
- Los métodos se declaran como sigue:

```
<tipo de retorno> <nombre del método>(
    <tipo parámetro 1> <parámetro 1>, …, <tipo parámetro n > <parámetro n >) {
        instrucción 1
        instrucción 2
        …..
}
```

Algunos ejemplos de métodos

Algunos ejemplos de métodos

- Dos métodos equivalentes:

```
int sumar (int num1, int num2) {
    int suma = 0
    suma = num1 + num2
    return(suma)
}
```

```
int sumar (int num1, int num2) {
    return(num1 + num2)
}
```

Algunos ejemplos de métodos

- Dos métodos equivalentes:

```
int sumar (int num1, int num2) {
    int suma = 0
    suma = num1 + num2
    return(suma)
}
```

```
int sumar (int num1, int num2) {
    return(num1 + num2)
}
```

- Un método vacío:

```
void imprimir_algo {
    System.out.println ("algo")
}
```

Clases

Clases

- Muy informalmente, se puede comparar una clase con una estructura.

Clases

- Muy informalmente, se puede comparar una clase con una estructura.
- El esquema general de declaración de una clase es el siguiente:

```
class <nombre clase> {
    <tipos de variable> <nombres de variable>;
    <tipos de método> <nombres de método> (<parámetros del método>){
        instrucción 1;
        instrucción 2;
        .....
    }
}
```

Clases

- Muy informalmente, se puede comparar una clase con una estructura.
- El esquema general de declaración de una clase es el siguiente:

```
class <nombre clase> {
    <tipos de variable> <nombres de variable>;
    <tipos de método> <nombres de método> (<parámetros del método>){
        instrucción 1;
        instrucción 2;
        .....
    }
}
```

- Por ejemplo:

```
class punto {
    int x;
    int y;
}
```

Clases

Clases

- El uso de las clases es similar al de los tipos de datos primitivos. Se utiliza la sintaxis siguiente:

```
<nombre clase> <nombre variable>;
```

Clases

- El uso de las clases es similar al de los tipos de datos primitivos. Se utiliza la sintaxis siguiente:

```
<nombre clase> <nombre variable>;
```

- Por ejemplo:

```
punto p;
```

Clases

- El uso de las clases es similar al de los tipos de datos primitivos. Se utiliza la sintaxis siguiente:

```
<nombre clase> <nombre variable>;
```

- Por ejemplo:

```
punto p;
```

- Antes de usar una variable de tipo clase, se le debe asignar memoria con la sentencia `new`:

```
<nombre variable> = new <nombre clase>;
```

Clases

- El uso de las clases es similar al de los tipos de datos primitivos. Se utiliza la sintaxis siguiente:

```
<nombre clase> <nombre variable>;
```

- Por ejemplo:

```
punto p;
```

- Antes de usar una variable de tipo clase, se le debe asignar memoria con la sentencia `new`:

```
<nombre variable> = new <nombre clase>;
```

- Por ejemplo:

```
p = new punto();
```

Clases

- El uso de las clases es similar al de los tipos de datos primitivos. Se utiliza la sintaxis siguiente:

```
<nombre clase> <nombre variable>;
```

- Por ejemplo:

```
punto p;
```

- Antes de usar una variable de tipo clase, se le debe asignar memoria con la sentencia `new`:

```
<nombre variable> = new <nombre clase>;
```

- Por ejemplo:

```
p = new punto();
```

- Se puede hacer todo de una vez:

```
punto p = new punto();
```

Clases

Clases

- A partir de aquí se puede usar la variable de tipo punto. Por ejemplo:

```
p.x = 5;
p.y = 2;
```

Paquetes

Paquetes

- Las bibliotecas de Java se llaman *paquetes*. Hay paquetes que pueden contener subpaquetes. Se utiliza la notación de punto para invocar métodos de los paquetes.

Paquetes

- Las bibliotecas de Java se llaman *paquetes*. Hay paquetes que pueden contener subpaquetes. Se utiliza la notación de punto para invocar métodos de los paquetes.
- Un paquete que utilizaremos casi siempre es el paquete `java.lang`, que contiene la clase `System`. Esta clase contiene a su vez el objeto `out`, que representa un espacio de salida (por defecto, la pantalla.) Este objeto ofrece, entre otros, los métodos `println` y `print`. Por ejemplo:

```
int n = 3;
char c = 'x';
System.out.println(n);
System.out.println(c);
System.out.println("Hola!");
```

Pasajes por valor y por referencia

Pasajes por valor y por referencia

- Las variables se pueden pasar a los métodos por *valor* o por *referencia*.

Pasajes por valor y por referencia

- Las variables se pueden pasar a los métodos por *valor* o por *referencia*.
- En el pasaje por valor, se pasa una copia del valor de la variable al método; el método trabaja sobre este valor; la variable original no sufre ninguna modificación.

Pasajes por valor y por referencia

- Las variables se pueden pasar a los métodos por *valor* o por *referencia*.
- En el pasaje por valor, se pasa una copia del valor de la variable al método; el método trabaja sobre este valor; la variable original no sufre ninguna modificación.

Pasajes por valor y por referencia

- Las variables se pueden pasar a los métodos por *valor* o por *referencia*.
- En el pasaje por valor, se pasa una copia del valor de la variable al método; el método trabaja sobre este valor; la variable original no sufre ninguna modificación.
- En el pasaje por referencia, se pasa la dirección de memoria de la variable al método; el método trabaja entonces directamente sobre la variable, que sufre los cambios que el llamado el método provoque.

Pasajes por valor y por referencia

- Las variables se pueden pasar a los métodos por *valor* o por *referencia*.
- En el pasaje por valor, se pasa una copia del valor de la variable al método; el método trabaja sobre este valor; la variable original no sufre ninguna modificación.
- En el pasaje por referencia, se pasa la dirección de memoria de la variable al método; el método trabaja entonces directamente sobre la variable, que sufre los cambios que el llamado el método provoque.
- Los tipos de datos primitivos se pasan en Java siempre por valor.

Pasajes por valor y por referencia

- Las variables se pueden pasar a los métodos por *valor* o por *referencia*.
- En el pasaje por valor, se pasa una copia del valor de la variable al método; el método trabaja sobre este valor; la variable original no sufre ninguna modificación.
- En el pasaje por referencia, se pasa la dirección de memoria de la variable al método; el método trabaja entonces directamente sobre la variable, que sufre los cambios que el llamado el método provoque.
- Los tipos de datos primitivos se pasan en Java siempre por valor.
- Los objetos se pasan en Java por referencia. La referencia se pasa por valor.

Pasajes por valor: un ejemplo

Pasajes por valor: un ejemplo

- Considere el siguiente ejemplo:

```
public static void cambiar(int  u) {
     System.out.println("Valor inicial de la variable u: " + u);
     u = u * 5;
     System.out.println("Valor final de la variable u: " + u);
}

public static void main(String[] args) {
     int u = 5;
     System.out.println("Valor de u antes del llamado: " + u);
     cambiar(u);
     System.out.println("Valor de u después del llamado: " + u);
}
```

Pasajes por valor: un ejemplo

- Considere el siguiente ejemplo:

```
public static void cambiar(int  u) {
     System.out.println("Valor inicial de la variable u: " + u);
     u = u * 5;
     System.out.println("Valor final de la variable u: " + u);
}

public static void main(String[] args) {
     int u = 5;
     System.out.println("Valor de u antes del llamado: " + u);
     cambiar(u);
     System.out.println("Valor de u después del llamado: " + u);
}
```

- La salida es:

```
Valor de u antes del llamado: 5;
 Valor inicial de la variable u: 5;
Valor final de la variable u: 25;
 Valor de u después del llamado: 5;
```

Pasajes por referencia: un ejemplo

Pasajes por referencia: un ejemplo

- Considere el siguiente ejemplo. Aquí creamos una clase *num* que sólo contiene un entero:

```
public static class num {
     int info;
}

public static void cambiar(num  u) {
     System.out.println("Valor inicial de la variable u: " + u.info);
     u.info = u.info * 5;
     System.out.println("Valor final de la variable u: " + u.info);
}

public static void main(String[] args) {
     num u = new num();
     u.info = 5;
     System.out.println("Valor de u antes del llamado: " + u.info);
     cambiar(u);
     System.out.println("Valor de u después del llamado: " + u.info);
}
```

Pasajes por referencia: un ejemplo

- Considere el siguiente ejemplo. Aquí creamos una clase *num* que sólo contiene un entero:

```
public static class num {
     int info;
}

public static void cambiar(num  u) {
     System.out.println("Valor inicial de la variable u: " + u.info);
     u.info = u.info * 5;
     System.out.println("Valor final de la variable u: " + u.info);
}

public static void main(String[] args) {
     num u = new num();
     u.info = 5;
     System.out.println("Valor de u antes del llamado: " + u.info);
     cambiar(u);
     System.out.println("Valor de u después del llamado: " + u.info);
}
```

- La salida es:

```
Valor de u antes del llamado: 5;
Valor inicial de la variable u: 5;
Valor final de la variable u: 25;
Valor de u después del llamado: 25;
```