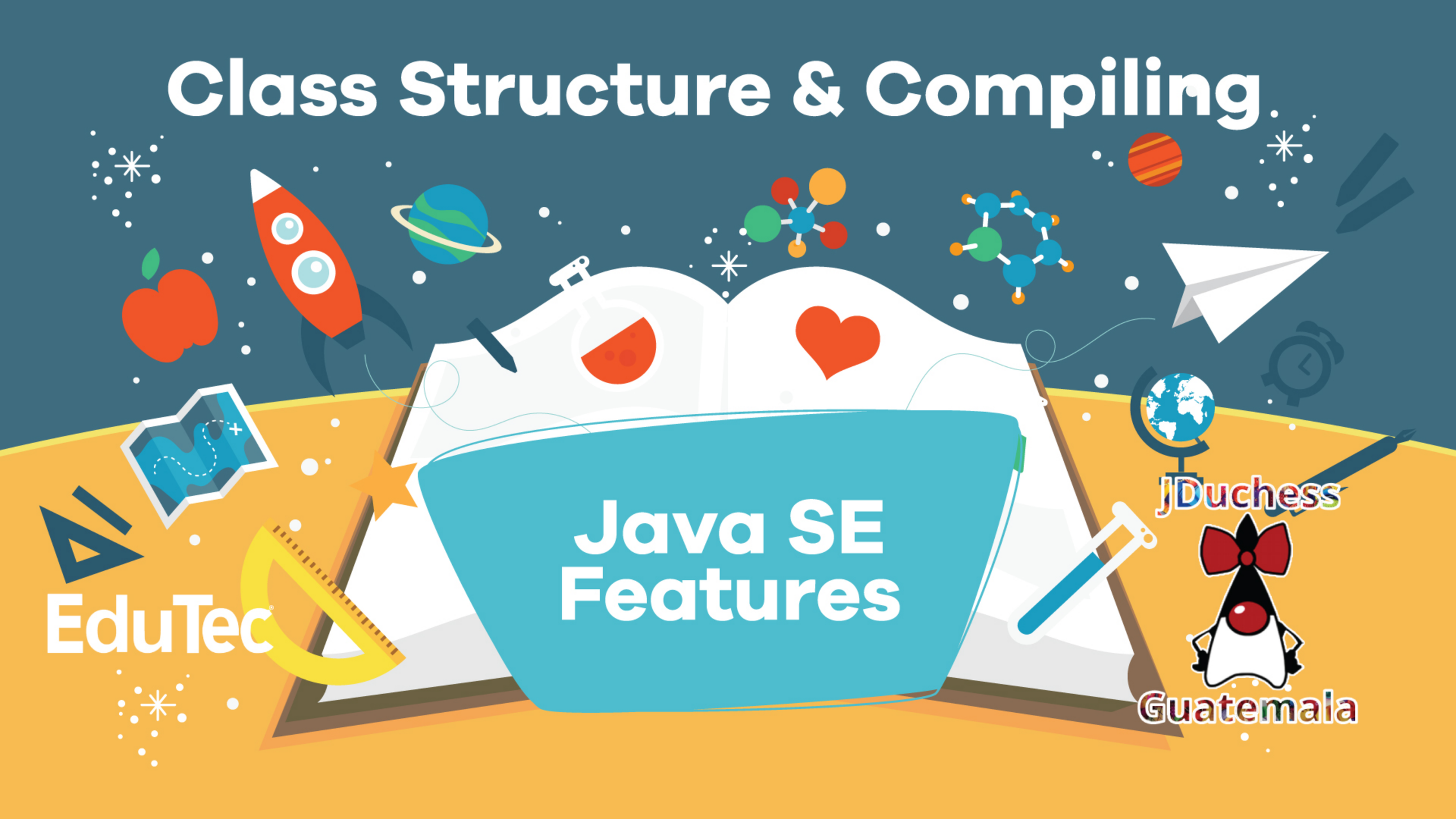
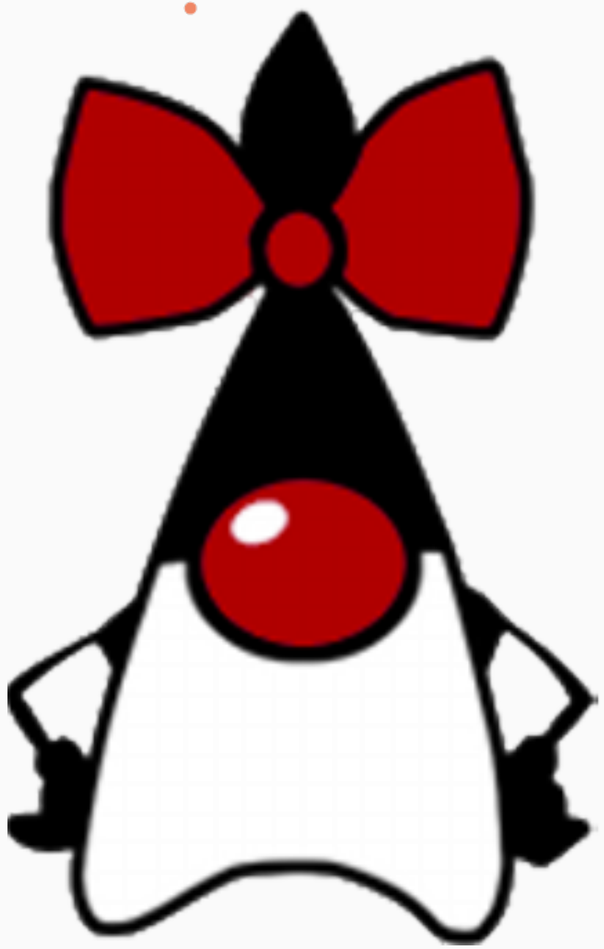
Class Structure & Compiling





JDuchess Guatemala

@itrjwyss

AMBASSADOR

Mercedes Wyss

@itrjwyss

Community Leader

Devs+502 & JDuchess Chapter Guatemala

Ex-JUG Member

Guatemala Java Users Group (GuateJUG)

Chief Technology Officer (CTO) at Produactivity

Full Stack Developer

Auth0 Ambassador &

Oracle Groundbreaker Ambassador

Duke's Choice Award

2016 Winner

Java™

AMBASSADOR

Oracle Groundbreaker Ambassador

Devs +502

Comunidad Desarrolladores en Tecnologías

< Google > en Guatemala

JDuchess

Guatemala

Class Structure

- package statement
- import statement
- comments
- class definition
  - Constants
  - Variable/fields
  - constructors
  - methods

@itrjwyss

Oracle
Groundbreaker
Ambassador

Paquetes

- Código Java está contenido en paquetes.
- Un paquete puede verse como un directorio.
- Podemos tener una jerarquía de paquetes.
- Siempre hay una paquete ***default***

package statement

- Provee información de en que paquete o jerarquía de paquetes se encuentra una clase.
- Si una clase no incluye paquete, está dentro del ***default***.

(jerarquía) paquetes

```
ClassStructure/
└── org
    └── jduchessgt
        └── Structure.java

2 directories, 1 file
```

@itrjwyss

package statement

```
package org.jduchessgt;
```


import statement

- Podemos incluir otras clases:
  - se encuentran en otros paquetes del proyecto.
  - librerías que estamos utilizando.

Estructura del Proyecto

```
ClassStructure/
└── org
    ├── jduchessgt
    │   └── Structure.java
    └── libraries
        └── Library.java

3 directories, 2 files
```

import statement

```
import org.libraries.Library;

import org.libraries.*;
```

@itrjwyss

Comentarios

- Son parte de nuestro código, pero no son código ejecutable.
- El compilador los ignora.
- Existen tres tipos:
    - Línea simple
    - Multiples líneas
    - Javadoc

Línea Simple

```
// This is a comment of a single line

//Initialized Field

//Non-Initialized Field

//Defining several variables of the same type.
```

@itrjwyss

Multiples Líneas

```
/* This is a multiple line comment
   We can have more than one line. */

/*
 * Sometimes multiple line comments use this
 * notation, to show them in a uniform way.
 */
```

Javadoc

- Comentarios especiales para crear la documentación del proyecto.
- Documentación es generada por un IDE como un conjunto de páginas HTML.
- Describen información importante sobre las clases, métodos o incluso las constantes.

Javadoc

```
/**
 * This is a Javadoc comment
 * @author Mercedes Wyss
 * @since version x.y.z
 * @param identifier we declare as many params as a method has
 * @return used in methods that return a value
 */
```

@itrjwyss

Oracle
Groundbreaker
Ambassador

class signature

- Está es la definición inicial de nuestra clase.
- Puede ser un poco compleja, ya que posee elementos obligatorios y opcionales.

```
[Access Modifiers] [Non-access Modifiers] class <Class Name>
[extends <Super Class Name>] [implements <Interface1 Name>,
<Interface2 Name>, ... <InterfaceN Name>] { }
```


class signature

- Estructura básica es:
  class keyword + nombre de la clase + llaves

```
class <nombre> { }
```

- Un archivo .java puede contener más de una clase definida.

class signature

- Al menos una clase está en nuestro archivo .java
- Para la clase que representa el archivo .java es mandatorio establecer un Access Modifier
- El archivo .java debe llamarse igual que la clase que representa

```
[Access Modifiers] class <Class Name> { }
```

```
public class Structure { }
```


Variables / Atributos

- Almacenan información que cambia a lo largo de la ejecución del programa.
- Al igual que en la Class Signature tenemos elementos que son obligatorios y opcionales.

```
[Access Modifier] <Data Type> <Identifier> [Initialization]
```

Variables / Atributos

```
//Initialized Field
String initialized = "Class Structure";

//Non-Initialized Field
double notInitialized;

//Defining several variables of the same type.
int var1, var2, var3;

//Applying Access Modifier
protected short accessModifier;
```

@itrjwyss

Variables / Atributos

- En Programación Orientada a Objetos manejamos el término ***atributo***. Son la representación de las características de un objeto.

- Concepto Variable lo utilizamos para las ***variables locales***, que son aquellas que definimos dentro de métodos, ciclos o estructuras de selección.

Variable Local

```
public double localVariable(){
    //Local Variable
    double variable = notInitialized * CONSTANT;
    return variable;
}
```

@itrjwyss

Constantes

- Son variables cuyo valor nunca va cambiar.
- Se deben inicializar al momento de declararse.
- Declaración

```
[Access Modifiers] final <Data Type> <Identifier> <Initialization>
```

```
final int CONSTANT = 10;
```

Métodos

- En otros lenguajes se les conoce como procedimientos y/o funciones.
- Definen el comportamiento de un objeto.
- La interacción de estos o con estos define la funcionalidad de toda la aplicación.

Métodos

- Su declaración posee elementos que son obligatorios y opcionales.

```
<Access Modifiers> [Non-access Modifiers | Specifier] <Return
Type> <Identifier> (<Parameters list>) [throws <Exception 1>,
<Exception 2>, ... <Exception N>] { <Method Body> }
```

Métodos

```
public double getNotInitialized(){
    return notInitialized;
}

public void setNotInitialized(double notInitialized){
    this.notInitialized = notInitialized;
}

public void withException() throws Exception {
    throw new Exception();
}
```

@itrjwyss

Constructores

- Métodos especiales que se ejecutan cuando se crea una instancia de un objeto.
- Conserva la misma definición que un método, solo que los constructores no tienen un tipo de dato de retorno, ni un Non-access Modifier.
- Otra característica es que se llaman igual que la clase.

```
<Access Modifiers> <Class Name> (<Parameters list>)
[throws <Exception 1>, <Exception 2>, ... <Exception N>]
{ <Constructor Body> }
```

Constructores

```
public Structure() throws Exception{
    this.notInitialized = 14.25;
    throw new Exception();
}

public Structure(double notInitialized){
    this.notInitialized = notInitialized;
}
```

@itrjwyss

Naming Conventions

Naming Conventions

- Java utiliza CamelCase combinada con otras reglas y convenciones.
- CamelCase está inspirada en los camellos. En palabras compuestas cada palabra empieza con letra mayúscula, simulando de está forma las jorobas de los camellos.
- Los nombres pueden contener letras, número y algunos caracteres especiales (dollar $ y guión bajo _)

Naming Conventions

- Los caracteres especiales pueden utilizarse en cualquier nombre, pero por convención para definir ***constantes***, ***variables*** / ***atributos***.
- Los nombres no pueden empezar con números.
- Hablaremos solo de las naming conventions relacionadas con los conceptos básicos de programación en Java (paquetes, constantes, variables/atributos y constantes)

paquetes

- Se escriben en minúsculas.
- Típicamente están definidos por una solo palabra, para casos especiales donde es una palabra compuesta se divide utilizando guión bajo (_).
- Una buen práctica para crear el paquete principal es utilizar el reverso del dominio web.
- Por ejemplo `org.jduchessgt.<subpackages>`

paquetes

```
org.jduchessgt.beans

org.jduchessgt.utils

org.jduchessgt.clients

org.jduchessgt.my_packeges
```


Clases

- Se definen con CamelCame, en palabras compuestas cada palabra empieza con mayúscula y el resto en minúsculas.
- Debe utilizarse sustantivos en singular
- Por ejemplo
  Comunidad, Meetup, Horario, Factura

Constantes

- Se definen en letras mayúsculas.
- Si son palabras compuestas se separan con guión bajo (_).
- Ejemplos:
  `DIA_REUNION, TIPO_CONFERENCIA, TIPO_TALLER`

Variables

- Se definen con Lower CamelCase, la primera palabra siempre empieza con minúscula, en palabras compuestas desde la segunda palabra se empieza con mayúscula.

- Es permitido utilizar los caracteres dollar ($) y guión bajo (_), incluso al inicio del nombre.

- Lo más importantes es utilizar nombres mnemónicos (representen lo que la variable almacena)

- Por ejemplo
  meetupDate, meetupPlace, $startDollar, name3

Main Method

@itrjwyss

Oracle
Groundbreaker
Ambassador

Método Main

- Sirve para crear una aplicación Java ejecutable.
- Se puede definir como la puerta de enlace entre el inicio del proceso Java ejecutado por la JVM (asignar memoria, tiempo de CPI, permite acceder a archivos) y nuestro código.
- Solo puede haber un único método main.

Método Main

- La definición del ***método main*** siempre es la misma, lo que varia suele ser la forma de colocar los elementos. Las reglas son:
    - Access modifier **public**
    - Non-access modifier **static**
    - El nombre debe ser *main*
    - El tipo de retorno es **void**
    - El argumento del método debe ser **String array**, o una variable argumento (**args**) de tipo **String**.

```
public static void main(String... args)

public static void main(String[] arguments)

public static void main(String[] HelloWorld)

public static void main(String[] args)

public static void main(String minnieMouse[])

public static void main(String[] args)

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

@itrjwyss

Algunos Comandos

Versión de Java

- Utilizamos el interprete Java, básicamente es el comando ***java***
- `java -version`

```
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
```

@itrjwyss

Compilar Java

- comando javac

```
javac [options] <classes>
```


Ejecutar

- Comando Java

```
java <class name> [argumentos]
```


https://github.com/itrjwyss/

https://www.facebook.com/itrjwyss

@itrjwyss