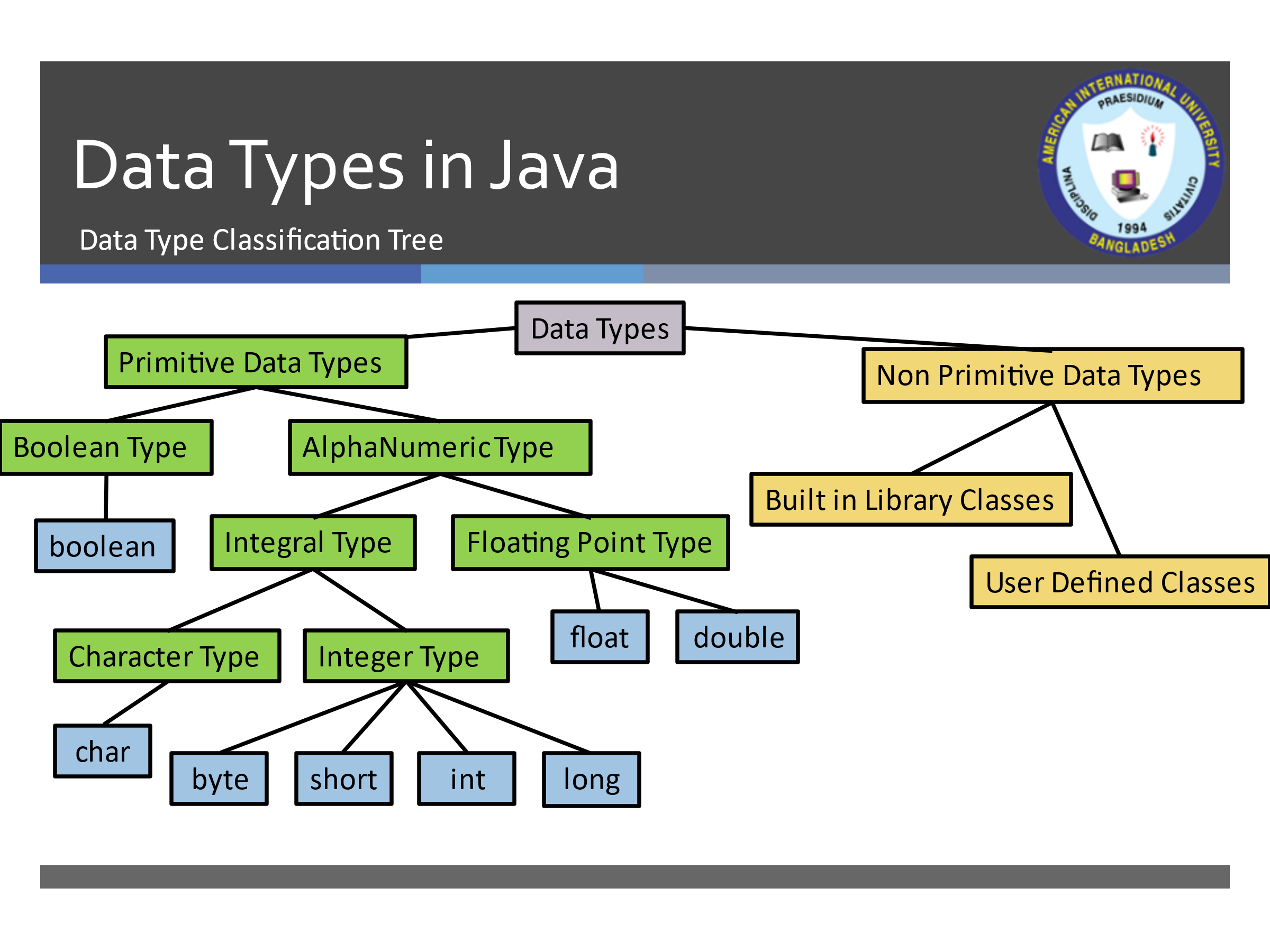
* Java was conceived by **James Gosling**, at Sun Microsystems, Inc. in 1991.
* It took 18 months to develop the first working version.
* This language was initially called **Oak** but was renamed **Java** in 1995.

The primary motivation was the need for a **platform-independent** language

* Real-world objects share two characteristics: They all have **state** and **behavior**.
* **Process-oriented model** was one of the earlier approaches to **Object-oriented programming**
* JVM(Java Virtual Machine) acts as a run-time engine to run Java applications.
* It converts Java bytecode into machines language

JVM is a part of JRE(Java Runtime Environment).

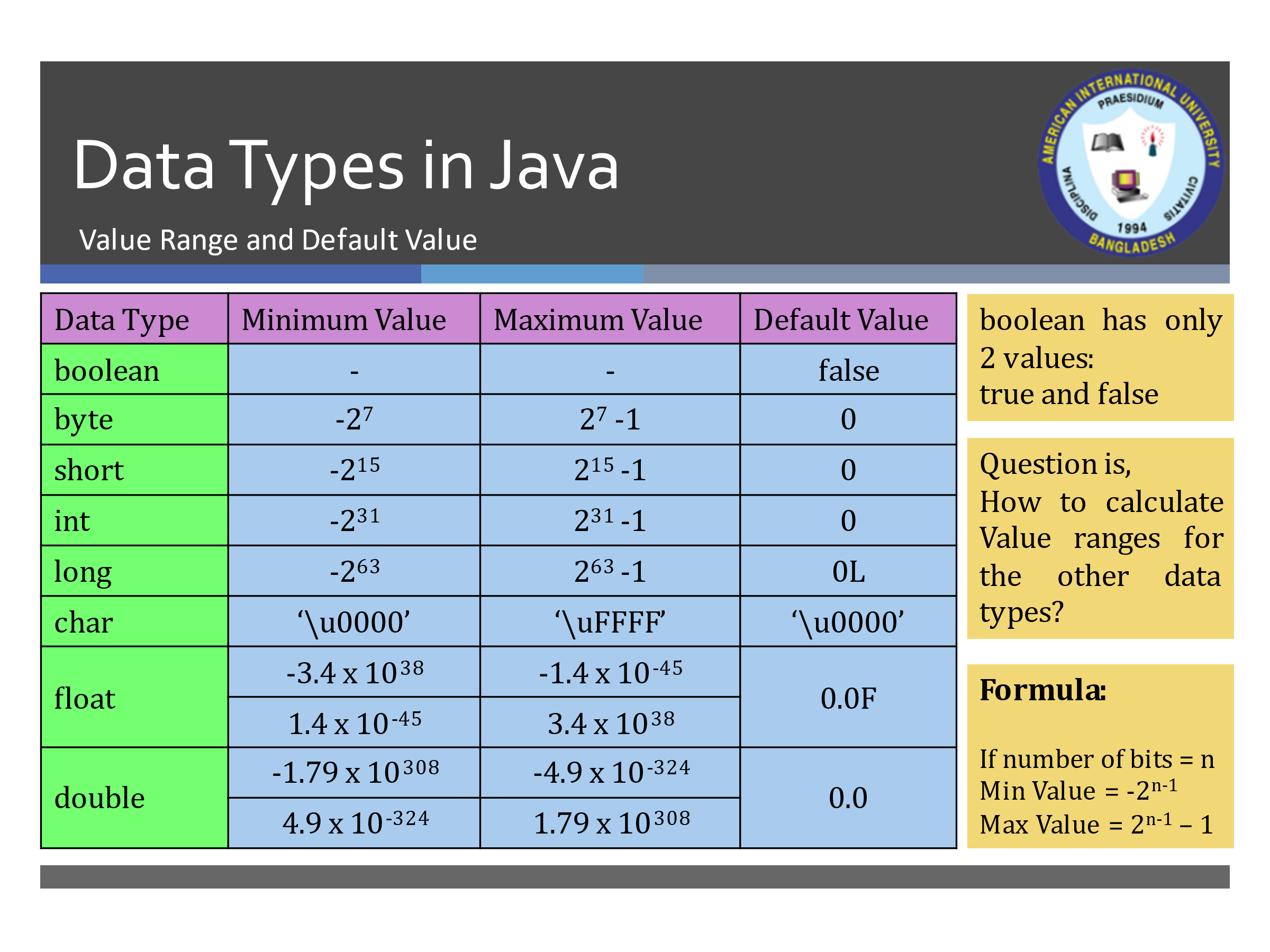
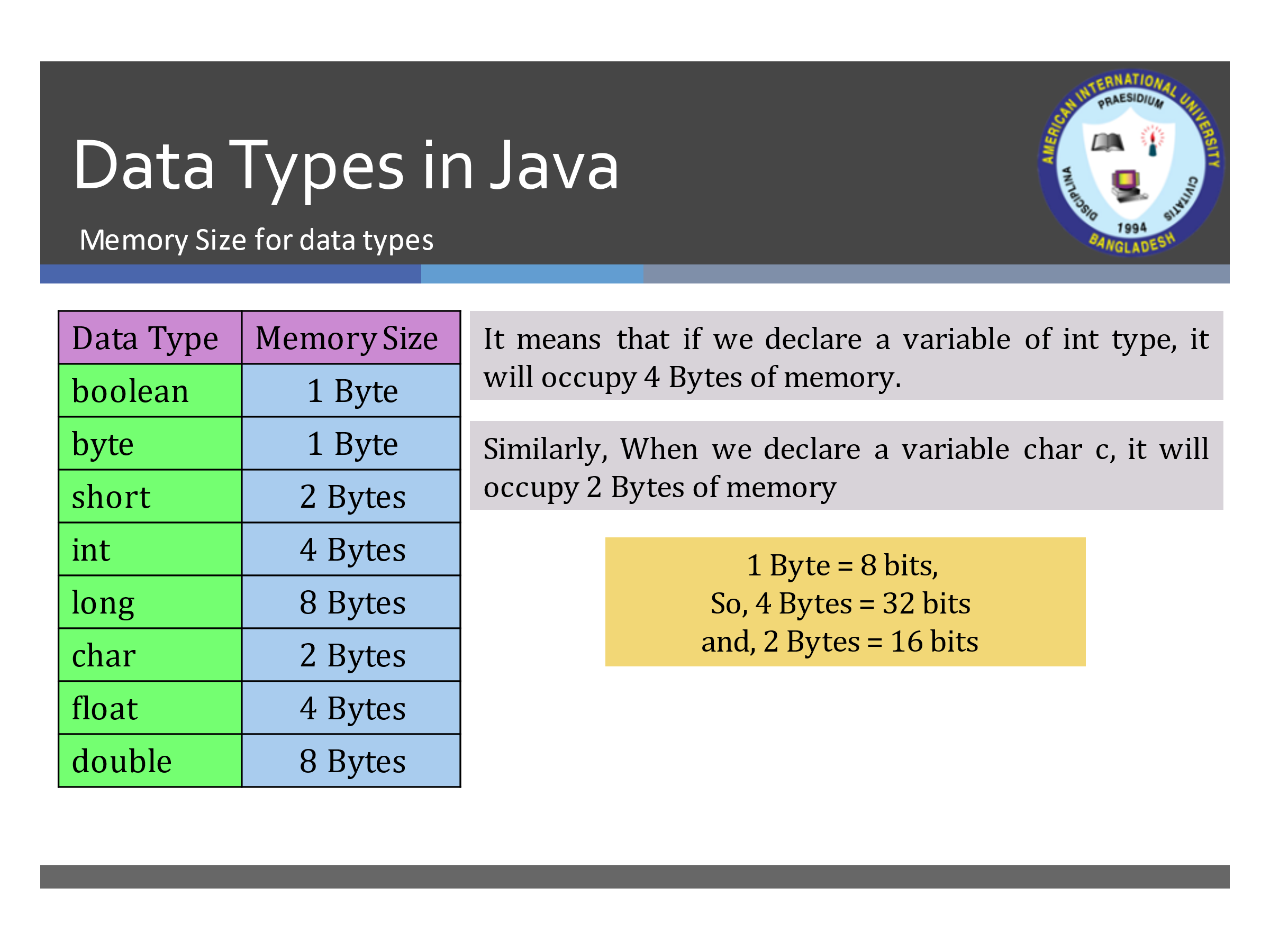


**Non-Primitive Data Types**

Any built in library classes in java and any classes that we will be creating are non primitive Data types. Some Examples:

1. String
2. System

3.Scanner



* Strings, which are widely used in Java programming, **are a sequence of characters**.
* In the Java programming language, **strings are objects**.
* String objects are handled specially by the compiler.
* String is the only class which has **implicit** **instantiation**.
* The String class is defined in the **java.lang package**.
* Strings are **immutable**. The value of a String object can never be changed.
* Scanner is found in the java.util package
* If the contents of a String have to be modified frequently, use the **StringBuffer** or **StringBuilder** class instead. StringBuffer and StringBuilder is **mutable**.
* However, String objects can be created **implicitly**.

String greetings = “Hello”;

* Classes are templates or blueprints for Objects
* An Object is the appearance of a class
* An object is an Instance of a class
* The process of creating an object is called instantiation
* The attributes of an object are called instance variables
* Variables can have one of 4 different visibilities:
  + public - the variable can be directly accessed from anywhere
  + private - the variable can only be directly accessed from within the class
  + protected - the variable can be access directly from within the class, within the package, or from within any subclass.
  + default (no modifier specified) - the variable can be accessed directly from within the package

**Local Variables**

* + Variables declared inside the scope of any method, constructor, loop, conditional statements are known as local variables.
  + Local variables do not have any existence outside the scope it is declared.
  + The concept of default value is not applicable for local variables.

**Class Variables**

* + Variables declared inside the scope of a class but outside the scope of any method or constructor are known as Class Variables if and only if the keyword static is used in the declaration of the variable.
  + It can be accessed from anywhere inside the class it is declared.
  + It can also be accessed from outside of the class it is declared depending its access modifiers.
  + We do not need any object to access it. It is accessed using the class name.
  + All the objects of a class, share the same memory for a class variable.

**Instance Variables**

* + Variables declared inside the scope of a class but outside the scope of any method or constructor are known as Instance Variables if and only if the keyword static is not used in the declaration of the variable.
  + It can be accessed from anywhere inside the class it is declared.
  + It can also be accessed from outside of the class it is declared depending its access modifiers.
  + We have to use an object to access it.
  + Each of the objects of a class, hold different memory for an instance variable.
* A static block is a normal block of code enclosed in braces, **{ }**, and preceded by the **static** keyword.
* A static block is also called **static initialization block**.
* The String class has many methods. The most used are:
  + length() - returns the number of characters in the String.
  + charAt() - returns the character at the specified index.
  + equals() - returns true if two strings have equal contents.
  + compareTo() -returns 0 if equal, Less than zero – if the invoking String is "less than” the other, Greater than zero - if the invoking String is "greater than" the other.
  + indexOf() - returns the index of specified String or character.
  + substring() -returns a portion of the String's text.
  + toUpperCase(), toLowerCase() - converts the String to upper- or lower-case characters.
* Encapsulation provides another important attribute: ***Access Control***
* There are two levels of access control:
  + At the top level—public, or package-private (no explicit modifier).
  + At the member level—public, private, protected, or package-private (no explicit modifier).
* At the top level—**public**, or **package-private** (no explicit modifier).
  + **public** 
    - A class may be declared with the modifier ***public***, in which case that class is visible to all classes everywhere.
  + **Package-private**
    - If a class has **no modifier** (the default, also known as **package-private**), it is visible only within its own package. (We will understand more about package-private, when we will study about packages).
* At the member level—**public**, **private**, **protected**, or **package-private** (no explicit modifier).
  + **public**
    - The public access modifier has the widest scope among all other access modifiers.
    - Methods or data members which are declared as public are accessible from every where in the program. There is no restriction on the scope of a public data members.
  + **private**:
    - The private modifier specifies that the member can only be accessed in its own class.
  + **protected**
    - The protected modifier specifies that the member can only be accessed within its own package (as with package-private).
    - In addition, by a subclass of its class in another package.
  + **package-private**:
    - The data members or methods which are not declared using any access modifiers i.e. having default access modifier are accessible only within the same package.
* The String class is defined in the **java.lang package**.

Definitions

A wrapper class is a java library class whose object wraps or contains a primitive data type.

Encapsulation is the mechanism that binds together code and the data it manipulates and keeps both safe from outside interference and misuse.

Inheritance is the mechanism of creating a new class by using an existing class.

Type Casting is the process of converting the value of one primitive data type to another primitive data type.

Tell the differences (Table format):

1.JVM ,JRE AND JDK

2.Implicit and Explicit type casting

3.Access specifier in java

4.Constructors and methods

5.Heap and stack memory

6.equals() and == in java

7.static(class) method and instance method

1. JVM, JRE, and JDK:

- JVM (Java Virtual Machine) is a (part of JRE) virtual machine that executes Java bytecode. It provides a runtime environment for Java applications, enabling them to be platform-independent.

- JRE (Java Runtime Environment) includes the JVM(a part of JDK), along with libraries and other files necessary for running Java applications. It does not include development tools such as compilers.

- JDK (Java Development Kit) is a software development kit that includes the JRE, along with development tools such as the Java compiler, debugger, and other utilities. It is used for developing Java applications.

JDK>JRE>JVM

2. Implicit and Explicit type casting:

- Implicit type casting, also known as widening or automatic type conversion, occurs when a value of a smaller data type is assigned to a variable of a larger data type. Java performs implicit casting when there is no loss of information or precision.

- Explicit type casting, also known as narrowing or manual type conversion, is required when assigning a value of a larger data type to a variable of a smaller data type. Explicit casting must be done explicitly by the programmer, as it may result in a loss of information or precision.

3. Access specifiers in Java:

- Access specifiers determine the accessibility or visibility of classes, methods, and variables in Java. There are four access specifiers in Java:

- Public: Accessible from anywhere, both within and outside the class.

- Protected: Accessible within the same package and subclasses (even if they are in a different package).

- Default (no specifier): Accessible within the same package only.

- Private: Accessible only within the same class.

4. Constructors and methods:

- Constructors are special methods used for initializing objects. They have the same name as the class and do not have a return type. Constructors are called automatically when an object is created using the "new" keyword.

- Methods, on the other hand, are functions defined within a class that perform specific tasks. They can have different names and return types. Methods are invoked explicitly by calling them using an object or class name.

5. Heap and stack memory:

- Heap memory is used for dynamic memory allocation and deallocation. It is where objects are stored in Java.

- Stack memory is used for storing method invocations and local variables. Each thread in Java has its own stack memory. Method calls and local variables are pushed onto the stack and removed when they are no longer needed.

6. equals() and == in Java:

- The "equals()" method is used to compare the content or values of two objects for equality. It is a method defined in the Object class and can be overridden by other classes to provide custom equality comparisons.

- The "==" operator is used to compare the references of two objects. It checks whether two object references point to the same memory location. It does not compare the content or values of the objects.

7. Static (class) method and instance method:

- Static methods, also known as class methods, belong to the class rather than any specific instance of the class. They can be called using the class name directly, without creating an instance of the class.

- Instance methods, on the other hand, are associated with instances of a class. They can access instance variables and are called on an object of the class using the dot operator.

THE IMPORTANCE OF OOP🡪

1. Encapsulation: OOP encapsulates data and methods within objects, providing data security and preventing data breach.
2. Code Organization: OOP promotes a structured approach to code organization, making it easier to understand and navigate through the codebase.
3. Code Reusability: OOP enables the creation of classes and objects that can be reused in different parts of the program or in other projects, saving development time and effort.
4. Inheritance: OOP supports inheritance, allowing classes to inherit properties and methods from other classes, promoting code reuse and reducing redundancy.
5. Polymorphism: OOP allows for polymorphism, where objects of different classes can be treated as objects of a common superclass, providing flexibility and extensibility in code design.

Setter Method:

A setter method is used to set or update the value of an attribute in an object.

It typically takes an argument (parameter) representing the new value to be assigned to the attribute.

It modifies the internal state of the object by assigning the new value to the attribute.

The method is usually named with the prefix "set" followed by the attribute name (e.g., setName, setAge).

It can include additional logic or validation to ensure that the assigned value is valid.

Getter Method:

A getter method is used to retrieve or access the value of an attribute from an object.

It usually does not take any arguments and returns the current value of the attribute.

It provides read-only access to the attribute without allowing direct modification.

The method is typically named with the prefix "get" followed by the attribute name (e.g., getName, getAge).

It does not modify the internal state of the object.

The purpose of using getter and setter methods is to encapsulate the internal state of an object and enforce controlled access to its attributes. By providing public getter and setter methods, you can maintain data integrity, apply validation rules, and hide the implementation details of the object's attributes from external code.

In Java, `System.out.println` and `System.out.print` are both methods used to display output to the console. The main difference between them lies in how they handle the printing of text:

1. System.out.println:

- The `println` method is short for "print line."

- It prints the specified text or value to the console and adds a newline character (`\n`) at the end.

- allowing subsequent output to appear on a new line.

- Example:

```java

System.out.println("Hello");

System.out.println("World");

```

Output:

```

Hello

World

```

2. System.out.print:

- The `print` method is used to print the specified text or value to the console without adding a newline character.

- It allows subsequent output to appear on the same line, right after the previously printed text.

- Example:

```java

System.out.print("Hello ");

System.out.print("World");

```

Output:

```

Hello World

```

In summary, `System.out.println` is useful when you want to print text or values and move the cursor to the beginning of the next line. `System.out.print` is suitable when you want to print text or values without moving the cursor to a new line, allowing for the next output to be displayed on the same line. The choice between the two depends on the desired formatting and presentation of the output.

In Java, inheritance is a mechanism that allows a class to inherit the properties and behaviors of another class. There are several types of inheritance, including single inheritance, multilevel inheritance, and hierarchical inheritance. Let's explore each of them:

1. Single Inheritance:

- Single inheritance occurs when a class inherits from only one superclass (parent class).

- It follows a one-to-one relationship between classes.

- The subclass (child class) inherits all the non-private members (fields and methods) of its superclass.

- Example:

```java

class Animal {

void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog is barking.");

}

}

```

In this example, the `Dog` class extends the `Animal` class, allowing the `Dog` class to inherit the `eat()` method from the `Animal` class.

2. Multilevel Inheritance:

- Multilevel inheritance occurs when a class inherits from a superclass, and that superclass itself inherits from another superclass.

- It follows a hierarchical inheritance chain.

- Each subclass inherits the members of its immediate superclass as well as all the members of the superclass hierarchy above it.

- Example:

```java

class Animal {

void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog is barking.");

}

}

class Labrador extends Dog {

void color() {

System.out.println("Labrador is brown.");

}

}

```

In this example, the `Labrador` class extends the `Dog` class, which, in turn, extends the `Animal` class. So, the `Labrador` class inherits both the `eat()` method from the `Animal` class and the `bark()` method from the `Dog` class.

3. Hierarchical Inheritance:

- Hierarchical inheritance occurs when multiple classes inherit from a single superclass.

- It forms a hierarchical structure or a tree-like structure of classes.

- Each subclass inherits the members of the superclass.

- Example:

```java

class Animal {

void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog is barking.");

}

}

class Cat extends Animal {

void meow() {

System.out.println("Cat is meowing.");

}

}

```

In this example, both the `Dog` class and the `Cat` class inherit the `eat()` method from the `Animal` class independently.

These are the three common types of inheritance in Java: single inheritance, multilevel inheritance, and hierarchical inheritance. Each type has its own characteristics and usage, and the choice of which one to use depends on the specific requirements and design of your program.

* Methods have the same visibility modifiers as variables
  + public - the method can be invoked from anywhere
  + private - the method can only be invoked from within the class
  + protected - the method can be invoked directly from within the class, within the package, or from within any subclass.
  + default (no modifier specified) - the method can be invoked directly from within the package
  + *// we will discuss in detail later.*