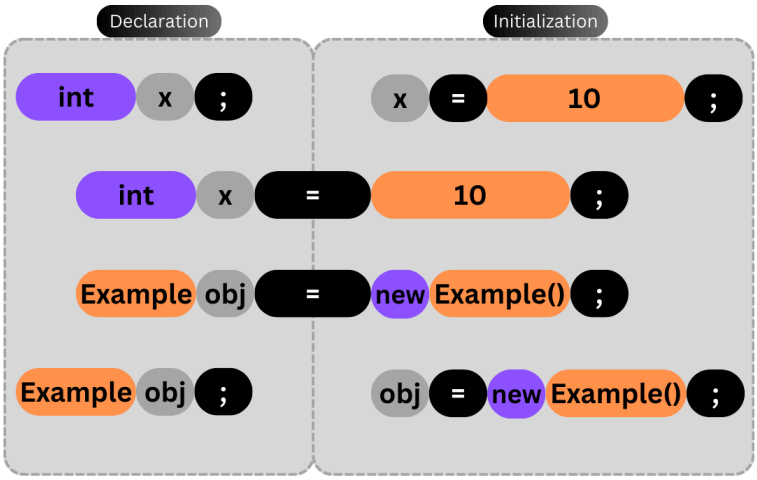
Variables

- A *variable* is an **identifier** associated with a memory location where data can be *stored, retrieved, and modified* during program execution.
    - Both value and object are considered data.
- **Literal**: A *constant* value assigned to a **variable**.
- A variable consists of two parts:
    - **Declaration**
    - **Initialization**



- *Declaration* is **mandatory**; *Initialization* is **optional**.
    - The **JVM** provides *default values* for uninitialized variables.

Default Value

- JVM-provided default values for uninitialized variables:

```
1  class DefaultValues {
2      byte byteVal;
3      short shortVal;
4      int intVal;
```

```
    long longVal;

    float floatVal;
    double doubleVal;

    char charVal;
    boolean boolVal;

    DefaultValues dv;
    String stringRef;
    System systemRef;

    public static void main(String[] args) {
        DefaultValues obj = new DefaultValues();
        System.out.println(obj.byteVal + "\n" + obj.shortVal + "\n" + obj.intVal + "\n" + obj.longVal);
        System.out.println(obj.floatVal + "\n" + obj.doubleVal);
        System.out.println(obj.charVal);
        System.out.println(obj.boolVal);

        System.out.println(obj.dv);
        System.out.println(obj.stringRef + "\n" + obj.systemRef);
    }
}
```

Types of Variables

- Variables are categorized based on **where** and **how** they are declared within the class:
    - **Non-Static Variable**
    - **Static Variable**
    - **Local Variable**

Non-Static Variable

- Values depend on the **object context**.
- Each object has a **separate copy**.
- Names of non-static variables **must be unique**.
- Initialization is **optional**; JVM provides default values.

Declaration

```
class NonStaticVariableExample {
    int instanceValue = 10; // Non-static variable
    public static void main(String[] args) {
    }
}
```

Access

- Accessed via **object reference**:

```
class NonStaticVariableAccess {
    int instanceValue = 10;
    public static void main(String[] args) {
        NonStaticVariableAccess obj1 = new NonStaticVariableAccess();
        NonStaticVariableAccess obj2 = new NonStaticVariableAccess();
        System.out.println(obj1.instanceValue + " " +
obj2.instanceValue);
        obj1.instanceValue = 99;
        obj2.instanceValue = -99;
        System.out.println(obj1.instanceValue + " " +
obj2.instanceValue);
    }
}
```

Storage Area

- Stored in **Heap Memory**.

Memory Allocation

- Occurs at **run-time**, after object creation.

Static Variable

- Value remains **common** across all objects.
- Each object shares a **single copy**.
- Static and non-static variables **cannot share the same name**.

- Initialization is **optional**; JVM provides default values.

Declaration

```
class StaticVariableExample {
    static int staticValue1 = 10;
    static int staticValue2 = 20;
    public static void main(String[] args) {
    }
}
```

Access

- Accessed in three ways:
    - **Directly**
    - Using **class name** (Recommended)
    - Using **object reference**

```
class StaticVariableAccess {
    static boolean staticFlag = true;
    public static void main(String[] args) {
        StaticVariableAccess obj = new StaticVariableAccess();
        System.out.println(staticFlag);                    // Direct
        System.out.println(StaticVariableAccess.staticFlag); // Class
name
        System.out.println(obj.staticFlag);                 // Object
reference
    }
}
```

Storage Area

- Initially in the **Method Area**, then moved to **Heap Memory**.

Memory Allocation

- Occurs at **class loading** time.

Local Variable

- Temporary variable; destroyed after the execution scope ends.
- Names of local and non-static/static variables **can be the same**, but **local variables have higher precedence**.
- Initialization is **mandatory** before use.

```
class ShadowingExample {
    static int globalValue = 100;
    public static void main(String[] args) {
        int globalValue = 10;
        System.out.println(globalValue);        // 10
        System.out.println(ShadowingExample.globalValue); // 100
    }
}
```

```
class LocalVariableDemo {
    public static void main(String[] args) {
        int x, y;
        y = 10;
        System.out.println(y);
    }
}
```

Declaration

- Declared **within methods or blocks** inside the class.

```
class LocalVariableDeclaration {
    public static void main(String[] args) {
        int localVar = 10; // Local Variable
    }
}
```

Access

- Accessed **directly**:

```java
class LocalVariableAccess {
    public static void main(String[] args) {
        int localValue = 10;
        System.out.println(localValue);
    }
}
```

Storage Area

- Stored in **Stack Memory**.

Memory Allocation

- Allocated **during method/code block execution**.
