**# Generate byte code in seperate directory**

1. To create a class file in same package

    javac -d . Shashwat.java

2. To create a class file in different package

    javac -d ../bin Shashwat.java

3. How to run a byte code from a different package?

    you need to set classpath of the directory where the class file resides.

    set classpath=../bin;

    java mypackage.Shashwat

4. Alternate way to set a class path

    java -classpath ../bin mypackage.Shashwat

**# Data Types ->**

**# Primitive data types - includes byte, short, int, long, float, double, boolean and char.**

A primitive data type specifies the size and type of variable values,

and it has no additional methods.

| Data Type | Size    | Range                                                    |

|-----------|---------|----------------------------------------------------------|

| byte      | 1 byte  | Stores whole numbers from -128 to 127                    |

| short     | 2 bytes | Stores whole numbers from -32,768 to 32,767              |

| int       | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647|

| long      | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807|

| float     | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits|

| double    | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits|

| boolean   | 1 bit   | Stores true or false values                             |

| char      | 2 bytes | Stores a single character/letter or ASCII values         |

**# Non-primitive data types - such as String, Arrays and Classes**

These are user-defined data types created by programmers.

Whenever a non-primitive data type is defined, it refers a memory location where the data is stored in heap memory i.e., it refers to the memory location where an object is placed. Therefore, a non-primitive data type variable is also called referenced data type or simply object reference variable.

1. Class

2. Object

3. String

4. Array

5. Interface

We will learn these as the course progress.

**# Constant in java**

A constant is a variable whose value cannot change once it has been assigned. Java doesn't have built-in support for constants.

In addition, a constant is cached by the JVM as well as our application, so using a constant can improve performance.

Add the keyword “final” in front of the variable declaration to make it a constant.

**# Variable in java**

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

It is a combination of "vary + able" which means its value can be changed.

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances. Unlike local variables,

we may use access specifiers for instance variables. If we do not specify any access specifier,

then the default access specifier will be used.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won’t halt the program. The compiler will replace the object name with the class name automatically

The general rules for naming variables are:

1. Names can contain letters, digits, underscores, and dollar signs

2. Names must begin with a letter

3. Names should start with a lowercase letter and it cannot contain whitespace

4. Names can also begin with $ and \_

5. Names are case sensitive ("myVar" and "myvar" are different variables)

6. Reserved words (like Java keywords, such as int or boolean) cannot be used as names

**# Literals in Java**

Literals in Java is a synthetic representation of boolean, numeric, character, or string data.

1. Integral literals

For Integral data types (byte, short, int, long), we can specify literals in 4 ways:-

a) Decimal literals (Base 10)

int x = 101;

b) Octal literals (Base 8)

int x = 0146; // should be prefix with 0.

c) Hexa-decimal literals (Base 16)

int x = 0X123Face; //The hexa-decimal number should be prefix with 0X or 0x.

Note: Although Java is case - sensitive but here 0X and 0x is treated as same (doglapan)

d) Binary literals (Base 2)

int x = 0b1111; // Literals value should be prefixed with 0b or 0B.

/

Note : By default, every literal is of int type, we can specify explicitly as long type by suffixed with l or L. There is no way to specify byte and short literals explicitly but indirectly we can specify. Whenever we are assigning integral literal to the byte variable and if the value is within the range of byte then the compiler treats it automatically as byte literals.

/

2. Floating-Point literal

For Floating-point data types, we can specify literals in only decimal form, and we cant specify in octal and Hexadecimal forms.

a) Decimal literals(Base 10)

double d = 123.456;

float a = 101.230f;

/

Note: By default, every floating-point literal is of double type, and hence we cant assign directly to the float variable. But we can specify floating-point literal as float type by suffixed with f or F. We can specify explicitly floating-point literal as double type by suffixed with d or D.

/

3. Char literals

For char data types, we can specify literals in 4 ways:

a) Single quote

char ch = 'a';

b) Char literal as Integral literal [ which represents the Unicode value of the character ]

char ch = 062;

c) Unicode Representation  allowed range is 0 to 65535.

char ch = '\u0061';

d) Escape Sequence [ A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler ]

char ch = '\n';

4. String literals

Any sequence of characters within double quotes is treated as String literals.

String s = "Hello";

5. Boolean literals

boolean b = true;

**# "Mixed Mode operation"**

mixing of character literals and integers in String concatenation operations

**# Java Keywords**

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

**# Widening or Automatic Type Conversion**

Widening conversion takes place when two data types are automatically converted. This happens when:

a. The two data types are compatible.

b. When we assign a value of a smaller data type to a bigger data type.

**# Narrowing or Explicit Conversion**

If we want to assign a value of a larger data type to a smaller data type we perform explicit type casting or narrowing.

This is useful for incompatible data types where automatic conversion cannot be done.

Here, the target type specifies the desired type to convert the specified value to.

  double d = 100.5467

  long l = (long)d;

Note:

Interesting fact:

int i = 257

b = (byte)i; [ b will be 1 (modulo 256) ]

**# Type Promotion in Expressions**

1. Implicit

While evaluating expressions, the intermediate value may exceed the range of operands and hence the expression value will be promoted. Some conditions for type promotion are:

Java automatically promotes each byte, short, or char operand to int when evaluating an expression.

If one operand is long, float or double the whole expression is promoted to long, float, or double respectively.

2. Explicit

Explicit Type Casting in Expressions

While evaluating expressions, the result is automatically updated to a larger data type of the operand. But if we store that result in any smaller data type it generates a compile-time error, due to which we need to typecast the result.

**# What is Automatic Type Promotion?**

The name Type Promotion specifies that a small size datatype can be promoted to a large size datatype. i.e., an Integer data type can be promoted to long, float, double, etc.

/

Note: This Automatic Type Promotion is done when any method which accepts a higher size data type argument is called with the smaller data type.

/

**# Unicode System**

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

Before Unicode, there were many language standards:

a. ASCII (American Standard Code for Information Interchange) for the United States.

b. ISO 8859-1 for Western European Language.

c. KOI-8 for Russian.

d. GB18030 and BIG-5 for chinese, and so on.

But, a particular code value corresponds to different letters in the various language standards.

The encodings for languages with large character sets have variable length.

Some common characters are encoded as single bytes, other require two or more byte.

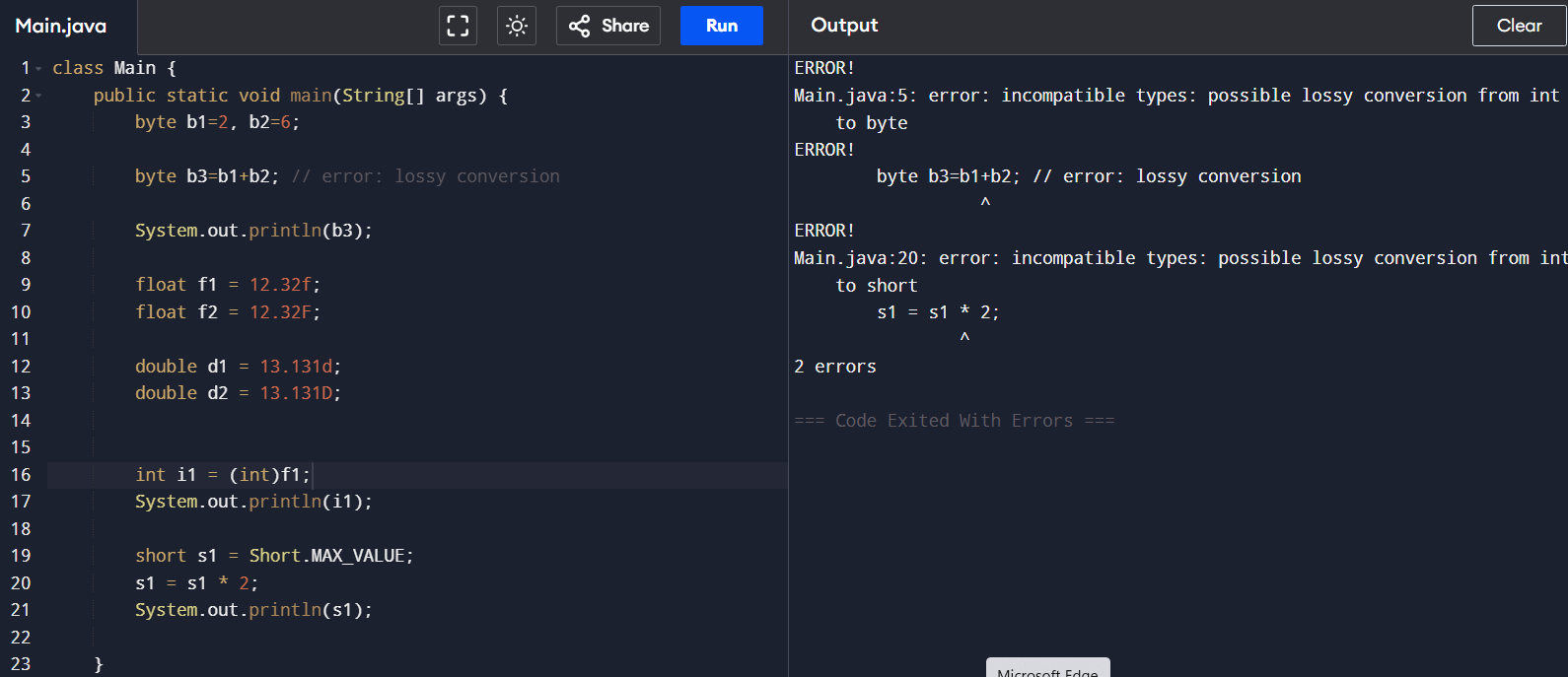
To solve these problems, a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.

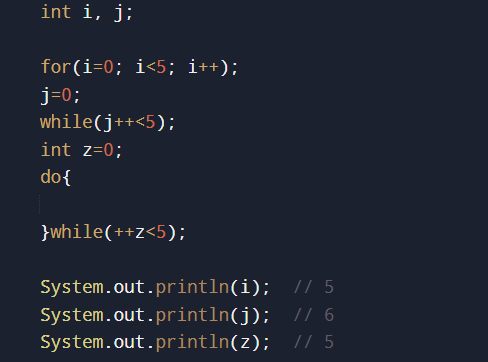
lowest value:\u0000

highest value:\uFFFF

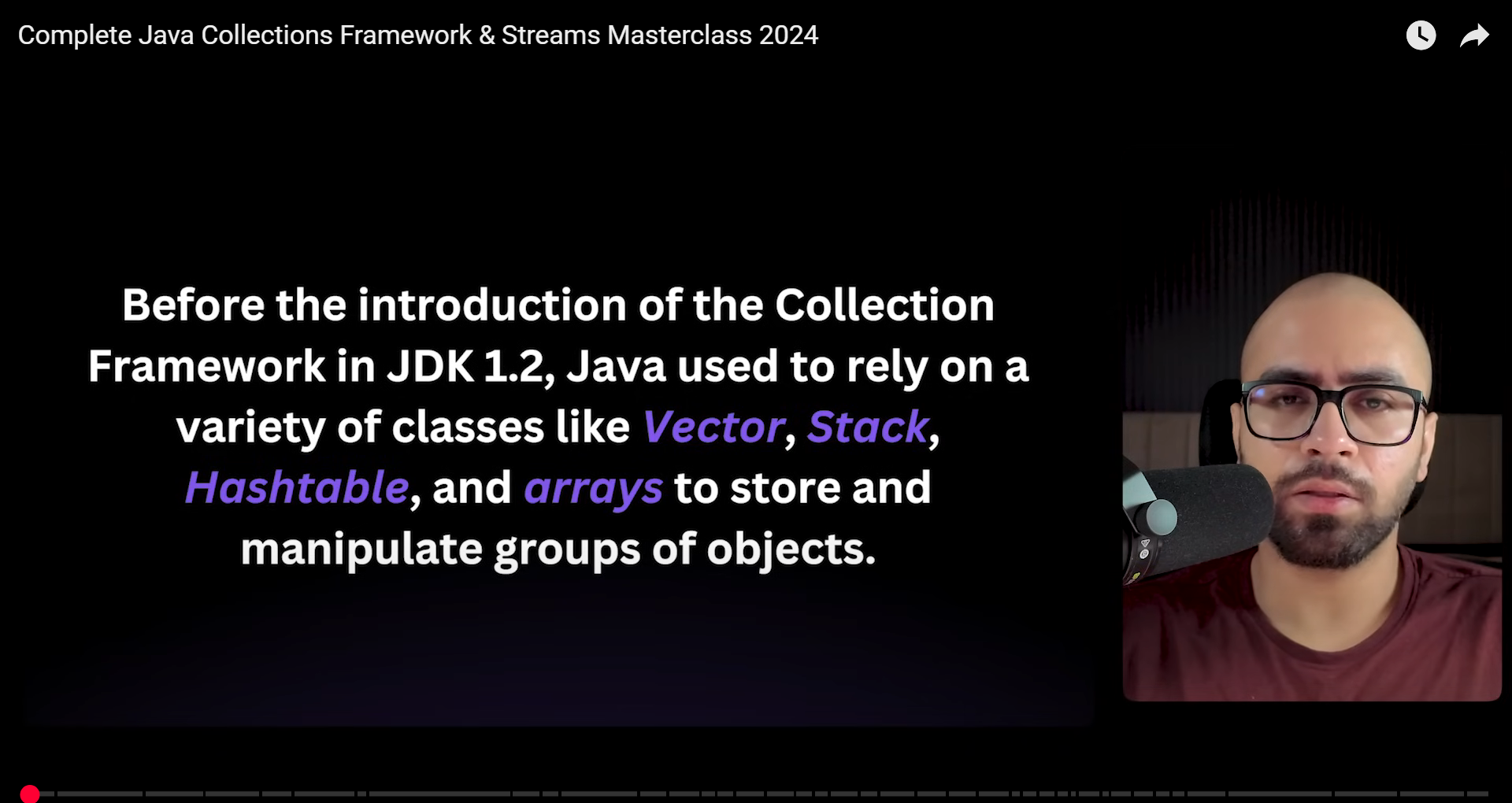
**Data Types** Questions



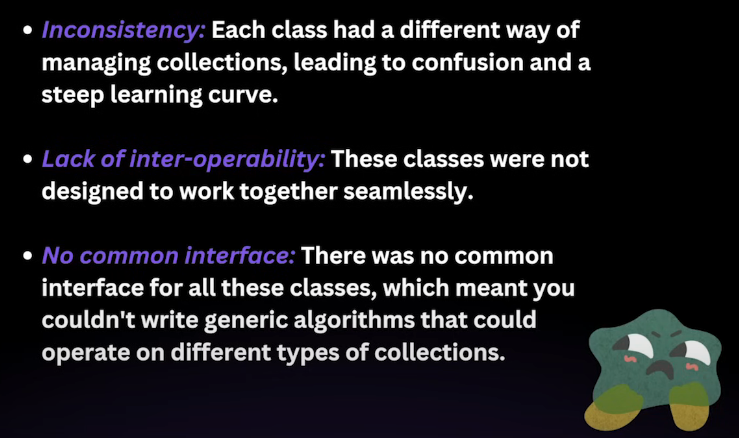
**Loops** Written in different way:



Collection Framework



Drawbacks:



Collection framework solves this:

