QUESTION:

Discuss compressively the java variable

ANSWER:

Variables in programming refer to a placeholder for value or a container for holding differnent types of data for performing differerent types of operations.

There are different ways in creating a variable which the method to be used is determined by the use of the data or the type of data to be saved in it.

To create a variable, one must specify the type and assign it a value:

Example is

```
type variableName = value;
```

firstly, one needs to understand the various types of data as it determines the type of function to be used to declare the variable.

Different types of data are:

Data types are divided into two groups:

Primitive data types: These includes byte, short, int, long, float, double, boolean, char.

Data Type	Size	Description
`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

Non-primitive data types are called reference types because they refer to objects.

The main difference between primitive and non-primitive data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

In Java programming language, variables are defines in different kinds:

- **Instance Variables:** Objects store their individual states in "non-static fields", that is, fields declared without the static keyword. Non-static fields are also known as instance variables because their values are unique to each instance of a class (to each object, in other words).
- **Class Variables:** A class variable is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated.
- **Local Variables:** Similar to how an object stores its state in fields, a method will often store its temporary state in local variables. The syntax for declaring a local variable is similar to declaring a field (for example, int count = 0;). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared — which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.

Parameters: Parameters are always considered as variable when they are initiated as arguments in a function or class. The important thing to remember is that parameters are always classified as "variables" not "fields". This applies to other parameter-accepting constructs as well (such as constructors and exception handlers)

If we are talking about "fields in general" (excluding local variables and parameters), we may simply say "fields". If the discussion applies to "all of the above", we may simply say "variables". If the context calls for a distinction, we will use specific terms (static field, local variables, etc.) as appropriate. One may also occasionally see the term "member" used as well. A type's fields, methods, and nested types are collectively called its members.

Naming Variables

Every programming language has its own set of rules and conventions for the kinds of names that you're allowed to use, and the Java programming language is no different. The rules and conventions for naming your variables can be summarized as follows:

- Variable names are case-sensitive.
- A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "$", or the underscore character "".
- For convention, is to always begin variable names with a letter, not "$" or "". Additionally, the dollar sign character, by convention, is never used at all. In some situations where auto-generated names will contain the dollar sign, but your variable names should always avoid using it. A similar convention exists for the underscore character; while it's technically legal to begin variable's name with "_", this practice is discouraged.
- White space within the variable name is not permitted.
- Subsequent characters may be letters, digits, dollar signs, or underscore characters. Conventions (and common sense) apply to this rule as well. When choosing a name for variables, it is good to use full words instead of cryptic abbreviations. Doing so will make ones code easier to read and understand. In many cases it will also make the code self-documenting; ‘loop_conditions’, ‘full_name’, ‘current_work’ are much more intuitive than abbreviated versions, such as s, c, and g.
- Variable name must not be a keyword or reserved word.
- By convention, If the name consists of only one word, it is good to spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word. The names gearRatio and currentGear are

prime examples of this convention. If your variable stores a constant value, such as static final int NUM_GEARS = 6, the convention changes slightly, capitalizing every letter and separating subsequent words with the underscore character. By convention, the underscore character is never used elsewhere.

Variable may be decleared and intiated at the same type or just decleared and then initiated inbetween the code.

Declearing a string

```
String name = "John";
```

Declearing an integer

```
int myNum = 15;
```