CPEN221 // Winter Session // 2020

# CPEN221 Principles of Software Construction: Section 1 Notes

**Author: Rudransh Kumar**

*Disclaimer*: These notes are an aggregation and integration of information from Professor Satish's prescribed readings, our classmates' comments, and fragments from various internet sources.

## Reading 2: The Need for Types

- **type**: a set of values and the operations permitted on the values.
    - *Example(s)*:
        - **boolean** is a type that has a binary set of values $\{0, 1\}$ and well-defined operations in the form of Boolean algebra: &&, ||, ^, !, etc.
        - **int** is a type that has a finite set of values $\mathbb{Z} \cap [2147483648, -2147483648]$ and well-defined operations in the form of (integer) arithmetic: +, -, $\times$, $\div$, etc.
- **primitive type**: a base type built into the programming language.
    - In Java, primitive types differ in that they are not objects; rather, they are stored as fixed values in memory.
    - *Example(s)*:
        - boolean, int, short, long, float, double, char, byte
- **non-primitive type**: a type built by a programmer, typically to model some real-word class of data items that may be aggregated by some common domain of possible values and permissible operations.
    - *Example(s)*:
        - String is a non-primitive type that has an infinite set of values containing all permutations of characters and well-defined operations such as concatenation, equivalence checking, lowercase transformation, etc.
- **user-defined type**: synonym for non-primitve type, with some exceptions
    - *Exception(s)*:

        - String is a type created by Java developers and shipped in the `java.lang` package, which supports "classes fundamental to the design of Java". Yet, it is neither considered a primitive type  because it is not stored as a fixed value in memory nor a user-defined type because it was made by the creators of the programming language.
- **class**: a blueprint for creating a new type, in which the class developer specifies the set of values that define instances of the type and operations permitted on them.
- **constructor method**: a method defined in the class definition which is called to initialize new objects of the type.
- **object**: a particular instance of a non-primitive type that has been declared.
- **declaration**: the process of identifying a new variable with an identifier (name) and type.
    - *Example(s)*:
        - `int number;`
        - `Date today;`
- **initialization**: the process of creating a new instance (object) of a class using the `new` keyword and by calling the class' constructor method.
    - *Examples(s)*
        - `jan_1_2018 = new Date(1, 1, 2018);`
- **reference**: a pointer that identifies the location of an object in memory.
    - Unlike pointers in C and C++, operations like referencing and dereferencing are not supported in Java.
- **developer (of a class)**: somone or some organization that creates a type
- **user (of a class)**: someone who uses a type
- What is the relationship between the developer and the user?
    - The developer aims to provide a meaningful type with useful operations, and in creating the type, they have an intimate understanding of how the type has been implemented.
    - The user aims to create objects of the type and perform operations in development of their own software.
    - The class specification defines the preconditions for the user to use the class and the postconditions the developer agrees to deliver, if the preoconditions are met.
- **defensive programming**: the implementation of programming practices aimed to ensure the continued functionality of some software even under unforeseen or unintended circumstances.
- **type safety**: a programming language's support for detecting errors related to the intermixing of types in a way that is either explicity disallowed or prohibitvely ill-defined.

- The type safety of a programming language is implemented collectively by the programming language's specification, which define the scope of legally permissable type intermixing, and the language's compiler that conducts static analysis in compile time.