

Structured Programming:

Structured programming is a programming paradigm that focuses on organizing code using well-defined control flow structures like sequences, conditionals (if-else), and loops. It aims to enhance code clarity and maintainability by reducing complexity and making it easier to understand. In structured programming, code is typically organized in a linear, sequential manner with the use of functions or procedures for modularity. Key principles include the avoidance of "goto" statements and promoting structured control flow.

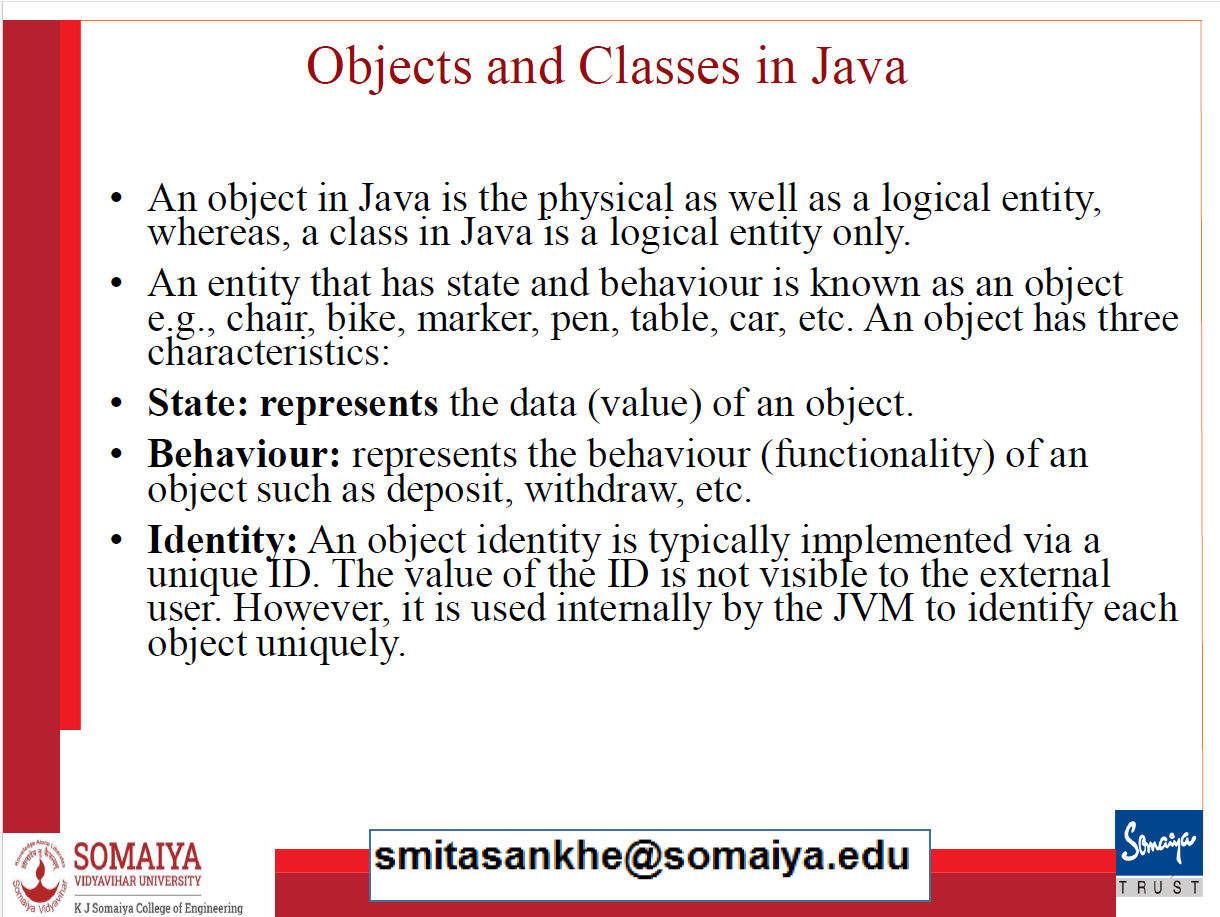
Procedural Programming:

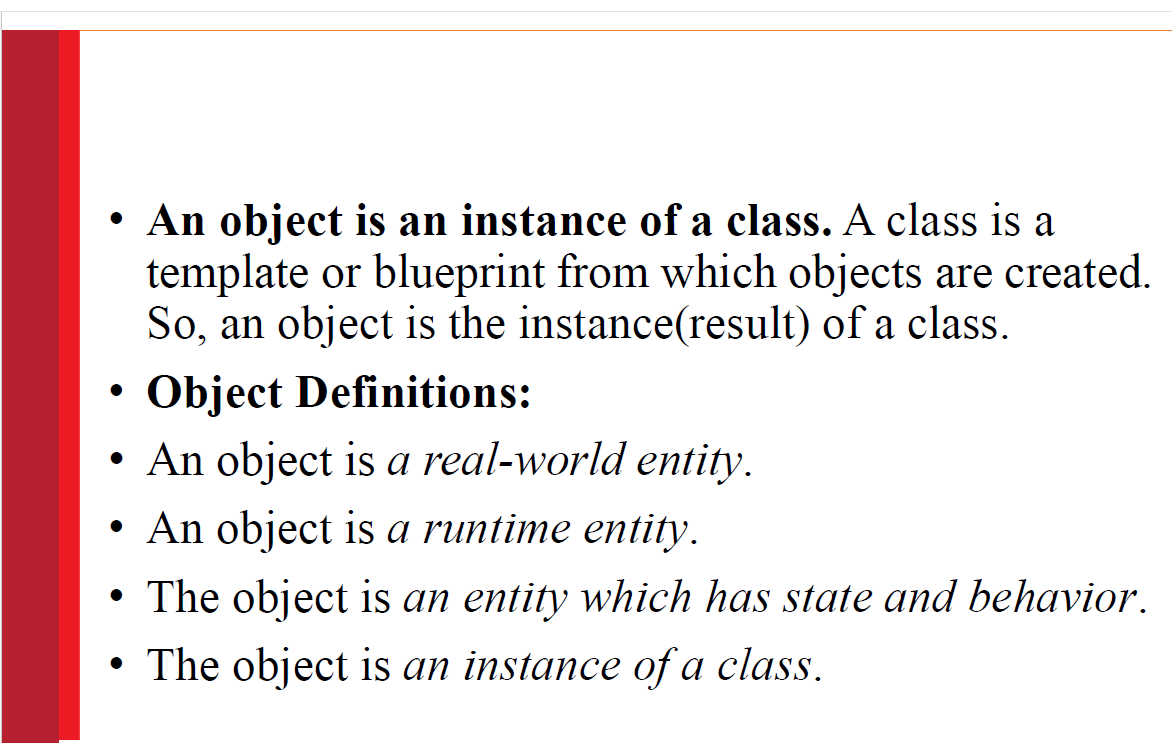
Procedural programming is a subset of structured programming that places a strong emphasis on breaking code into procedures or functions. It promotes code modularity, top-down design, and better code reusability. In procedural programming, code is organized around procedures or functions that encapsulate specific tasks. Variables are typically local to these procedures, minimizing unintended side effects. Procedural programming languages like C and Pascal are known for their use of procedures and functions to structure code.

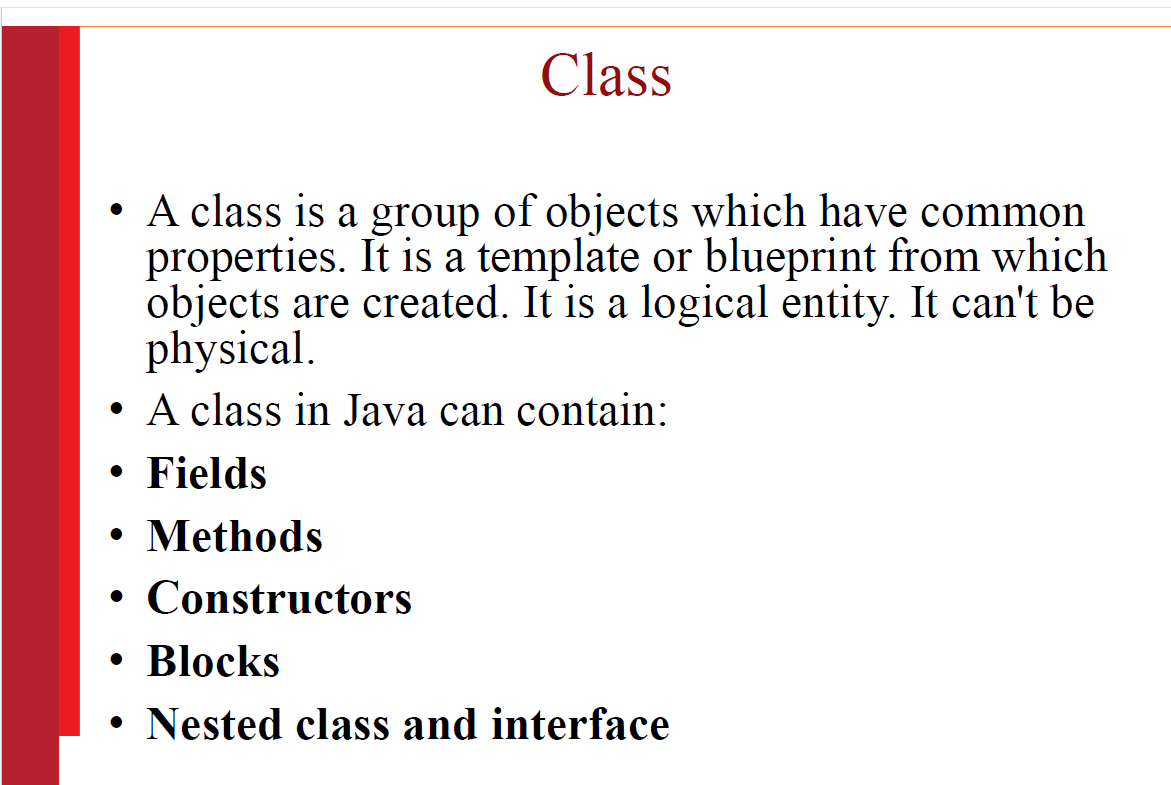
Object-Oriented Programming (OOP):

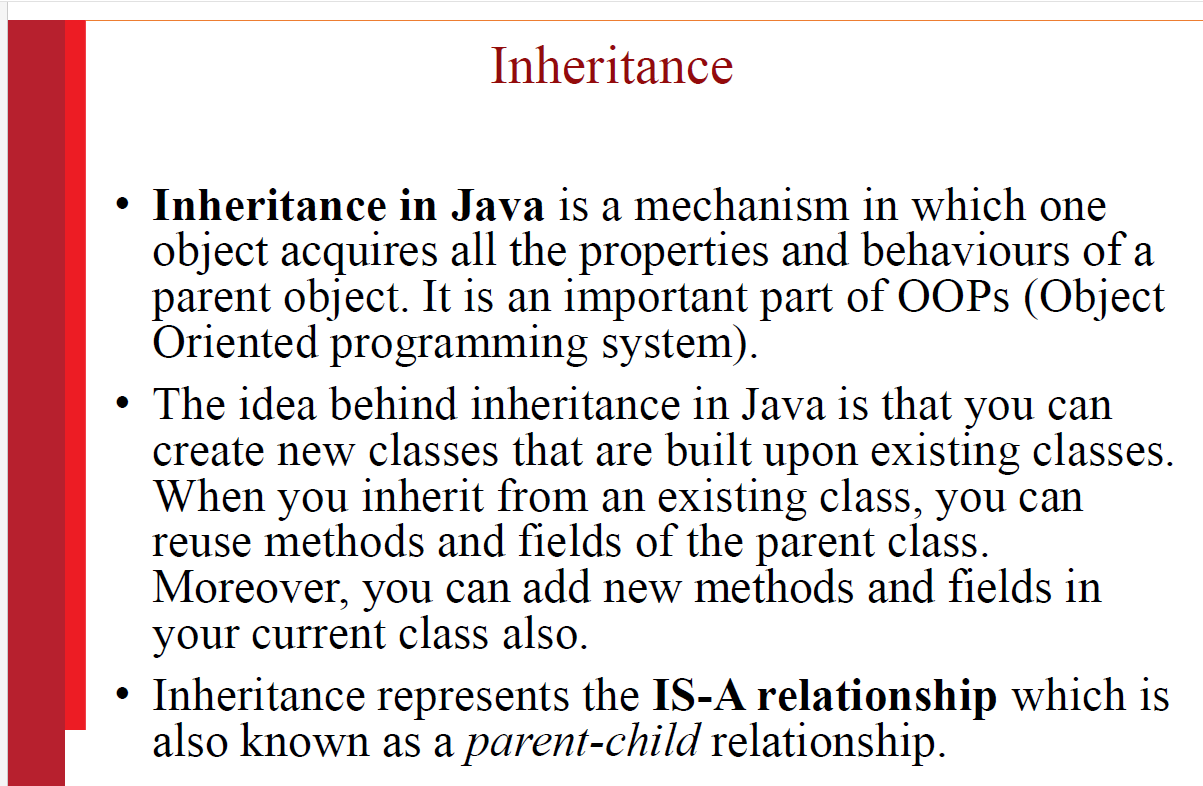
Object-oriented programming is a programming paradigm that models software as a collection of objects, each representing a real-world entity with data (attributes) and behavior (methods/functions). OOP promotes code organization around these objects, enabling code reusability through inheritance and creating a hierarchy of related objects. Key principles of OOP include encapsulation (hiding internal details), inheritance (sharing attributes and methods among objects), and polymorphism (objects responding differently to the same method call based on their class). Common OOP languages include Java, C++, and Python.

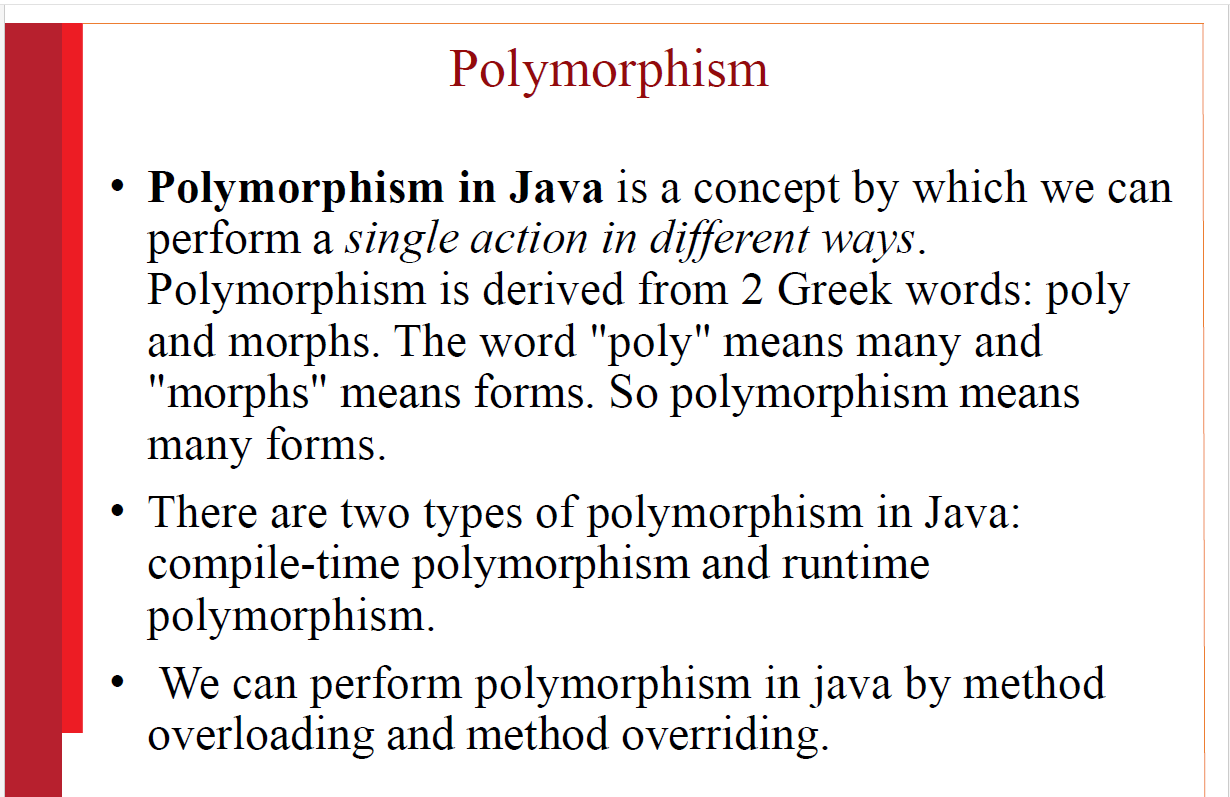
In summary, structured programming focuses on control flow structures, procedural programming adds the concept of procedures/functions for modularity, and object-oriented programming models software as a collection of objects with data and behavior, promoting code organization around real-world entities and fostering code reusability. Each paradigm has its own advantages and is suitable for different types of applications and problem-solving approaches.

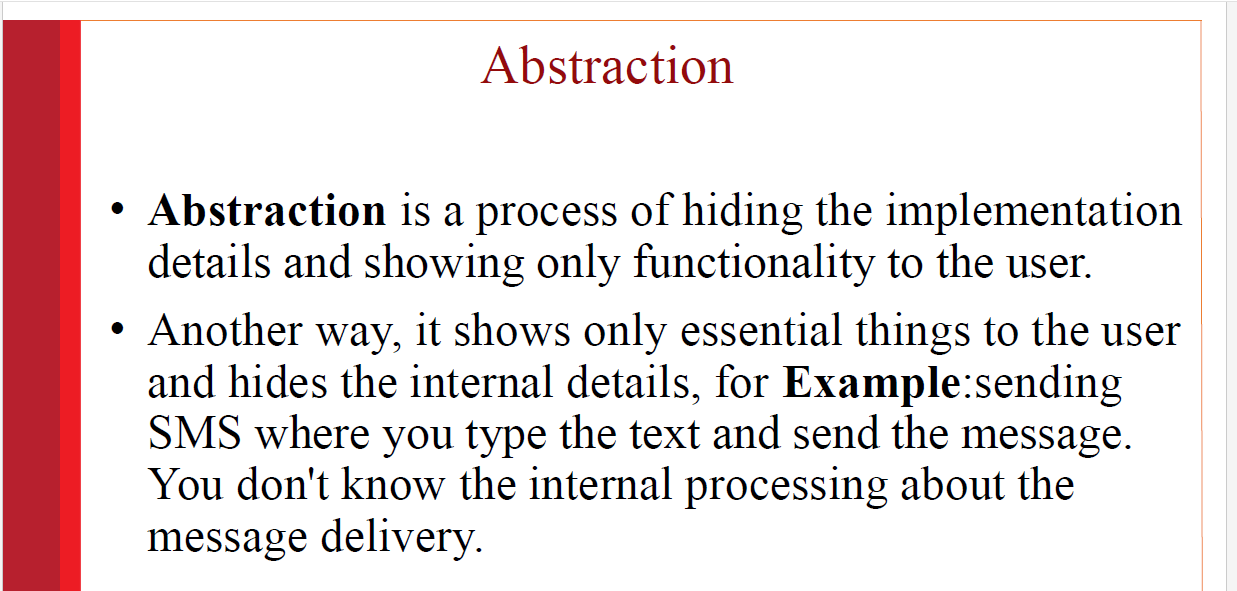


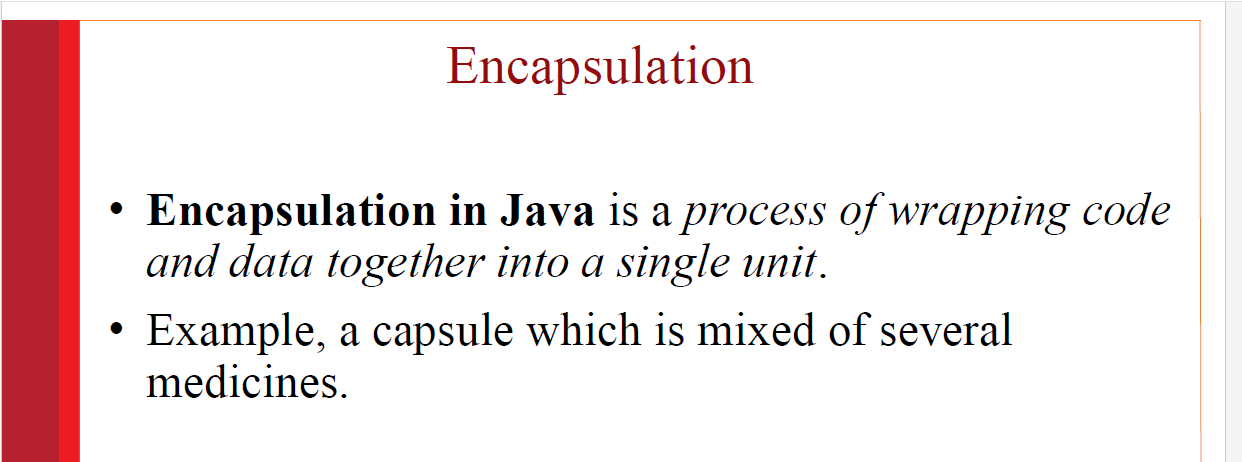












Platform Independent

•Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines

•while Java is **a write once, run anywhere language**. A platform is the hardware or software environment in which a program runs.

Secured

•Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

•**No explicit pointer**

•**Java Programs run inside a virtual machine**

Robust

•Robust simply means strong. Java is robust because:

•It uses strong memory management.

•There is a lack of pointers that avoids security problems.

•There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral

•Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

Portable

•Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation

High-performance

•Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++).

•Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++,

Distributed

•Java is distributed because it facilitates users to create distributed applications in Java.

Multi-threaded

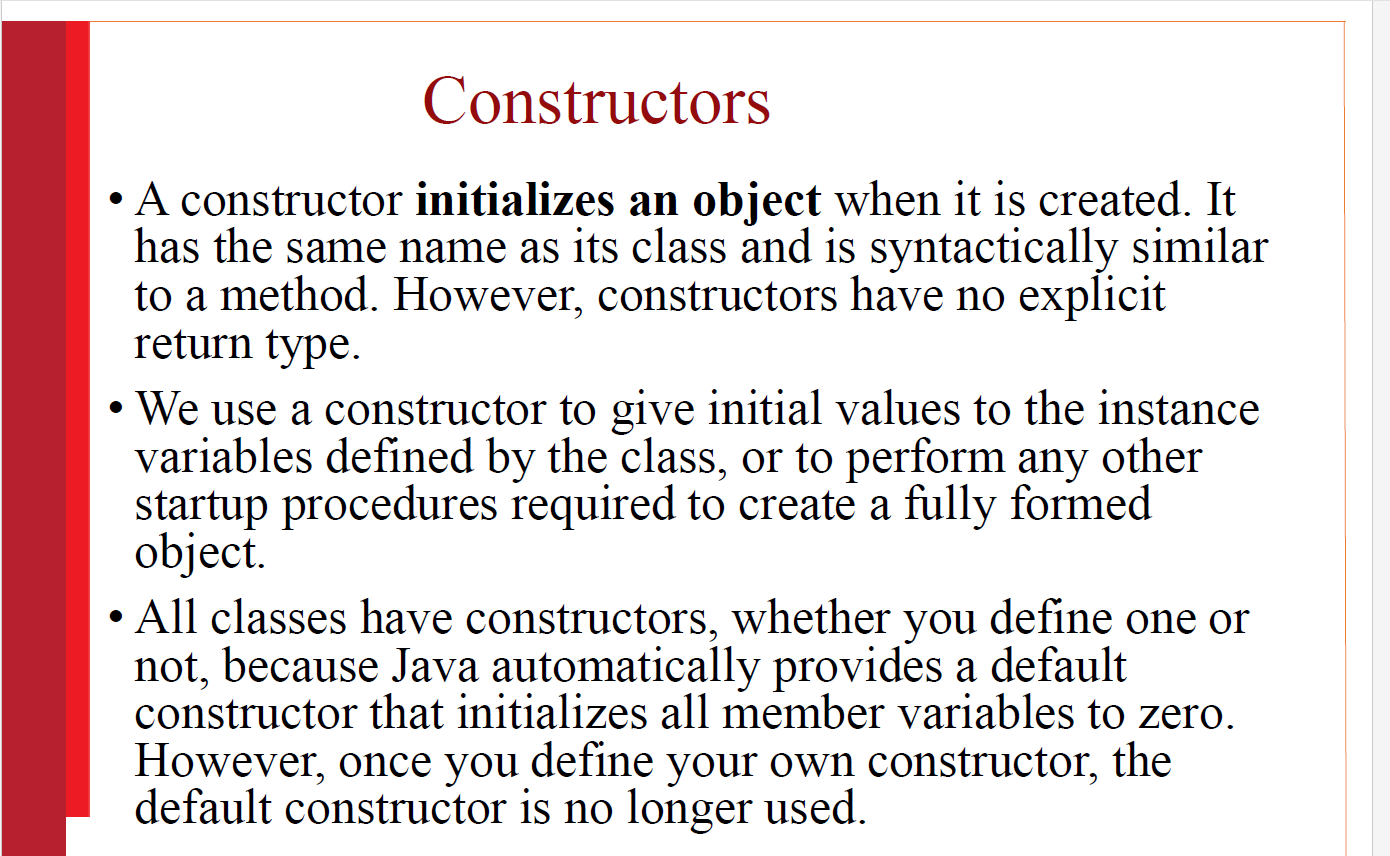
•A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.

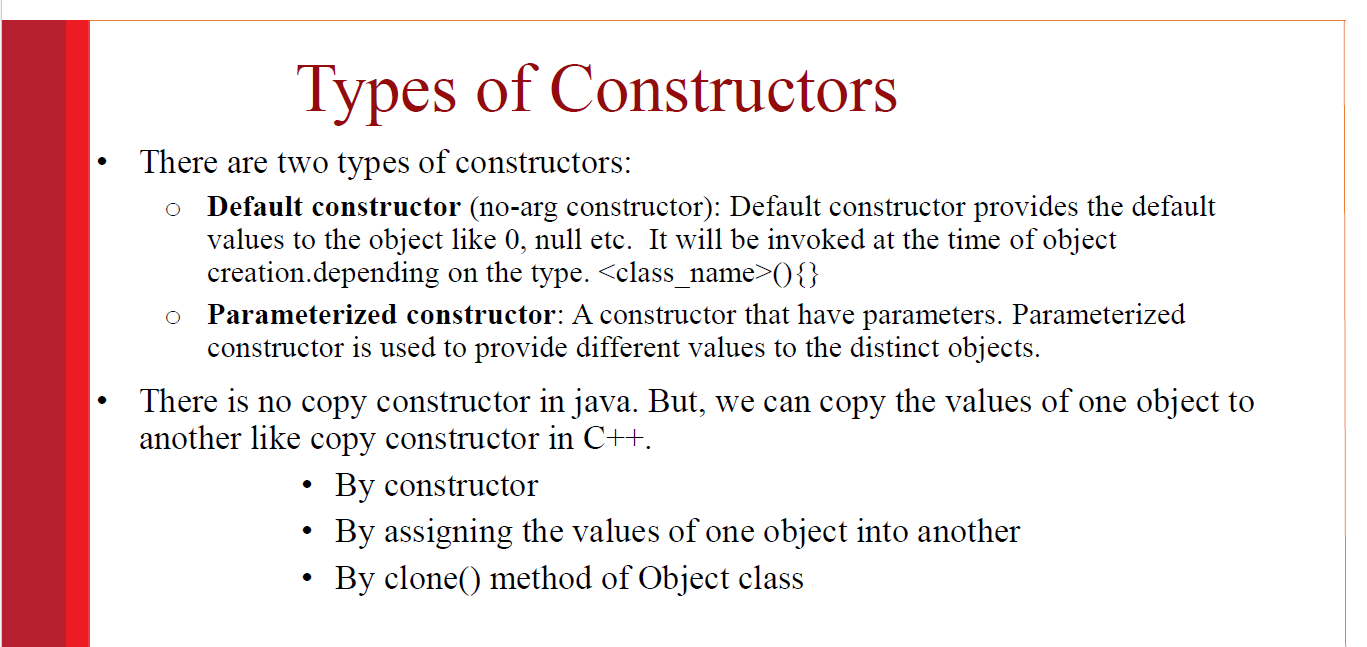
•The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

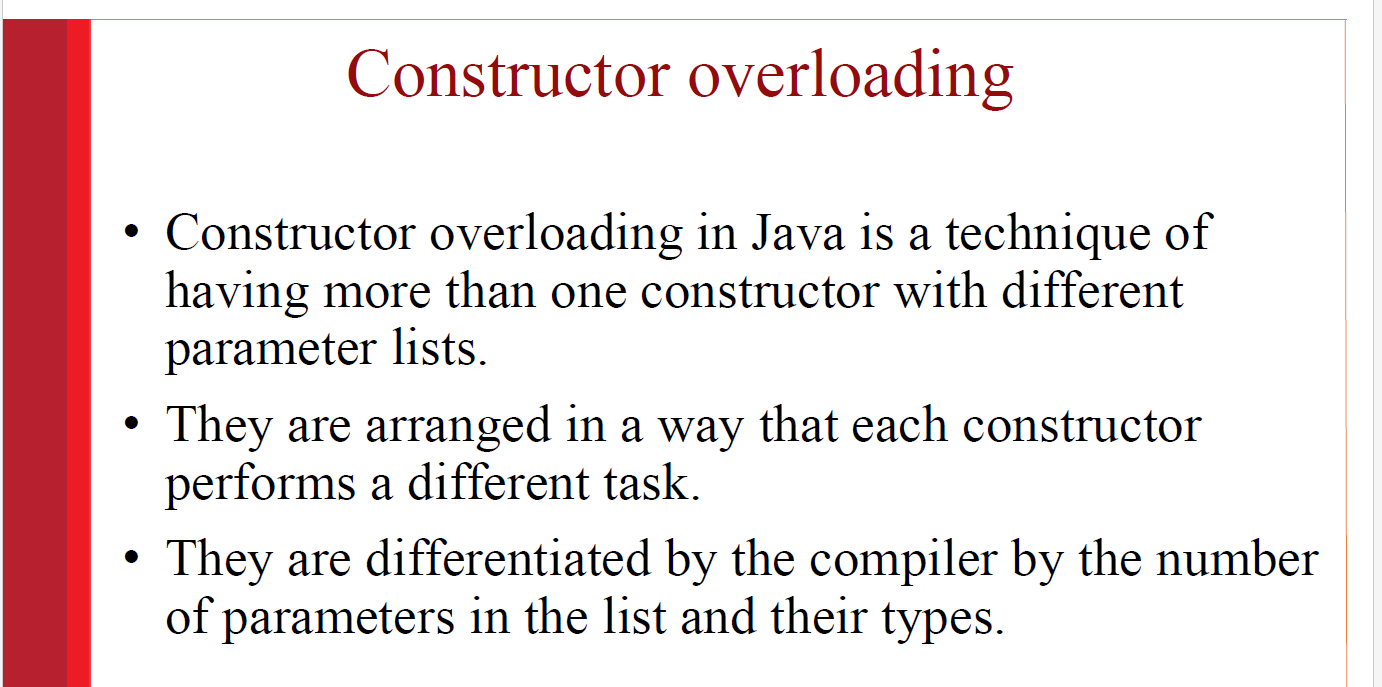
Dynamic

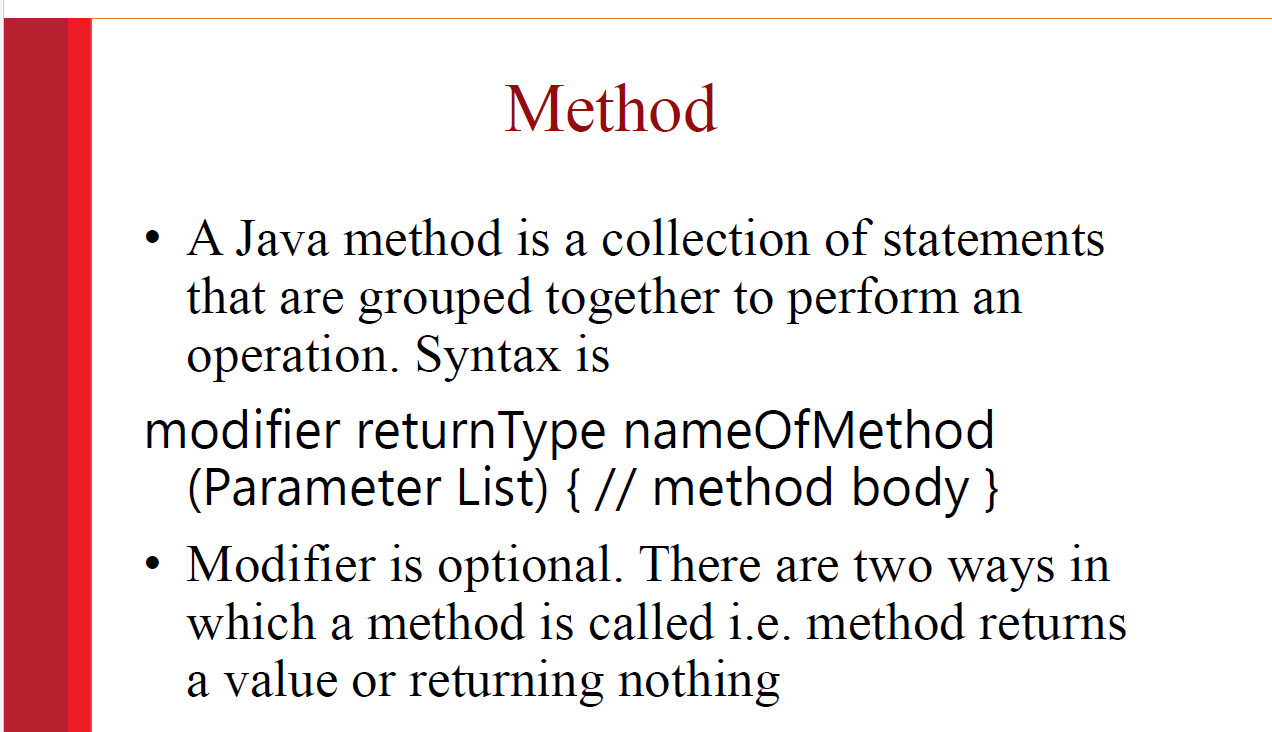
•Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

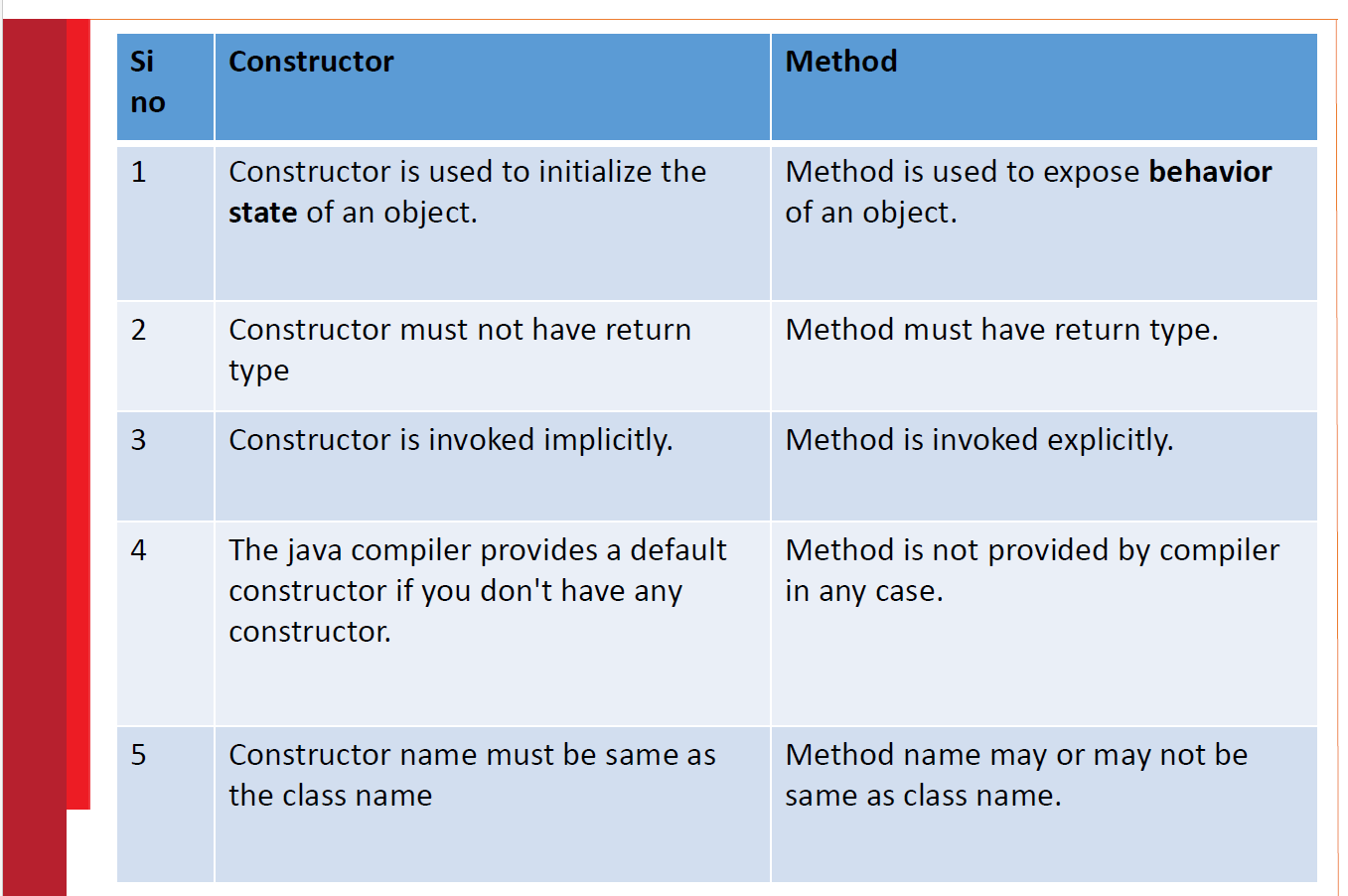
•Java supports dynamic compilation and automatic memory management (garbage collection)

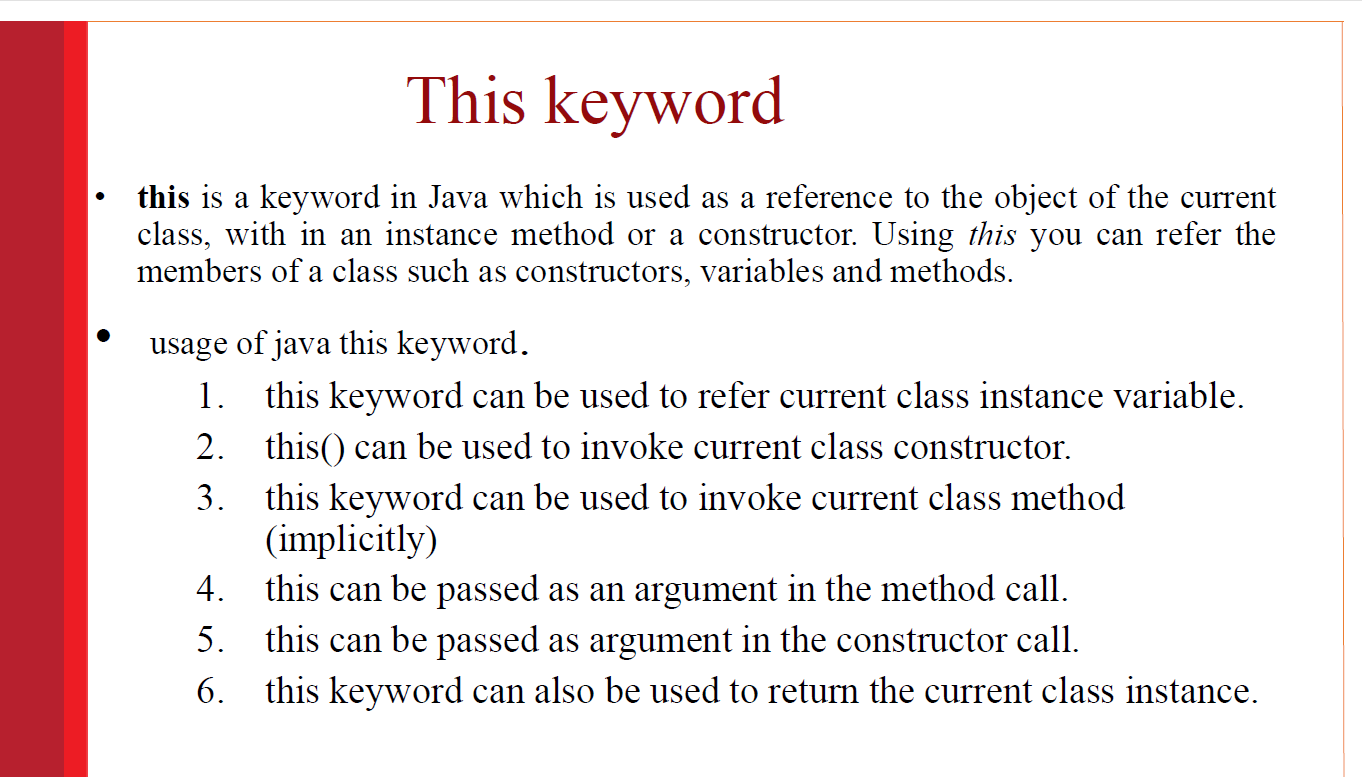


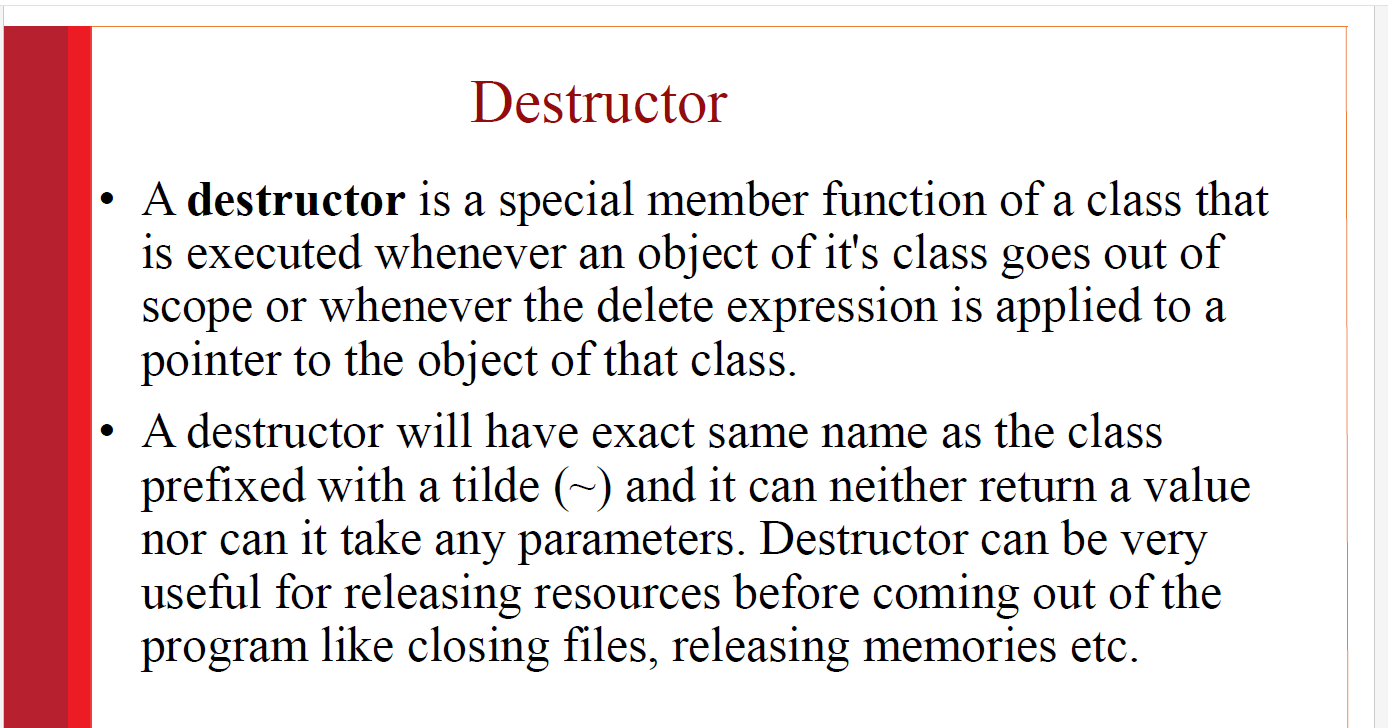


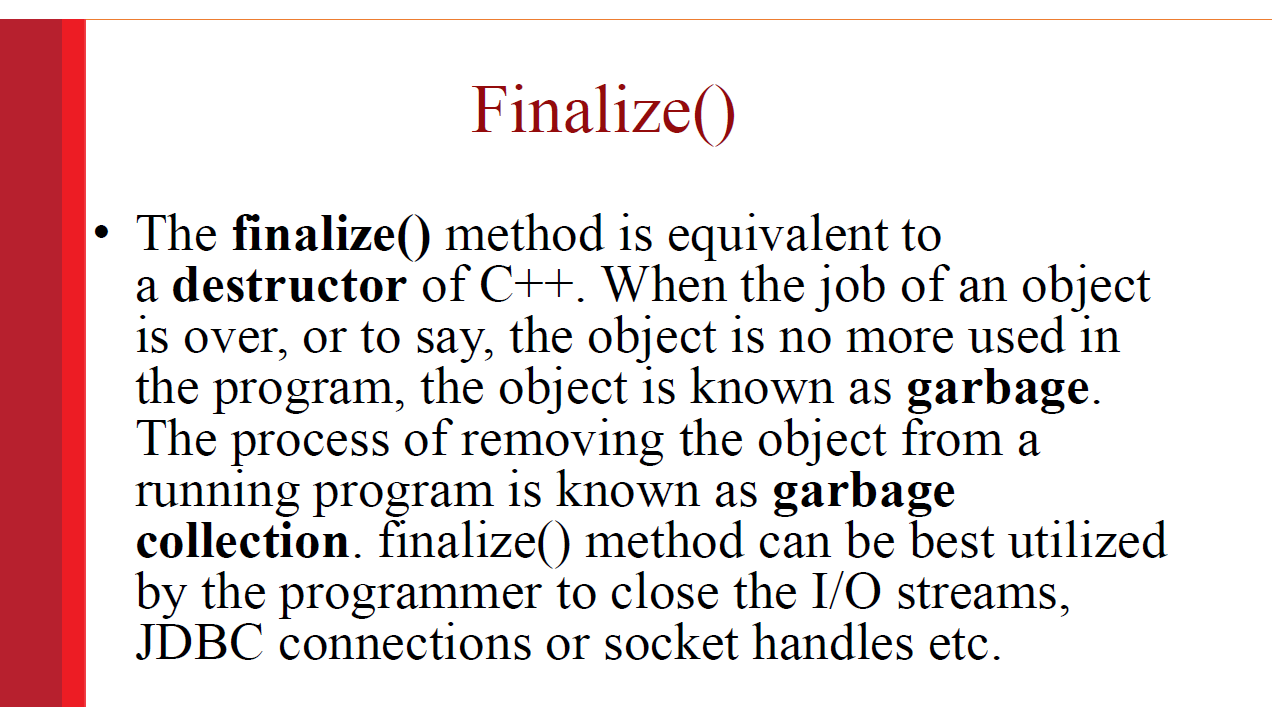


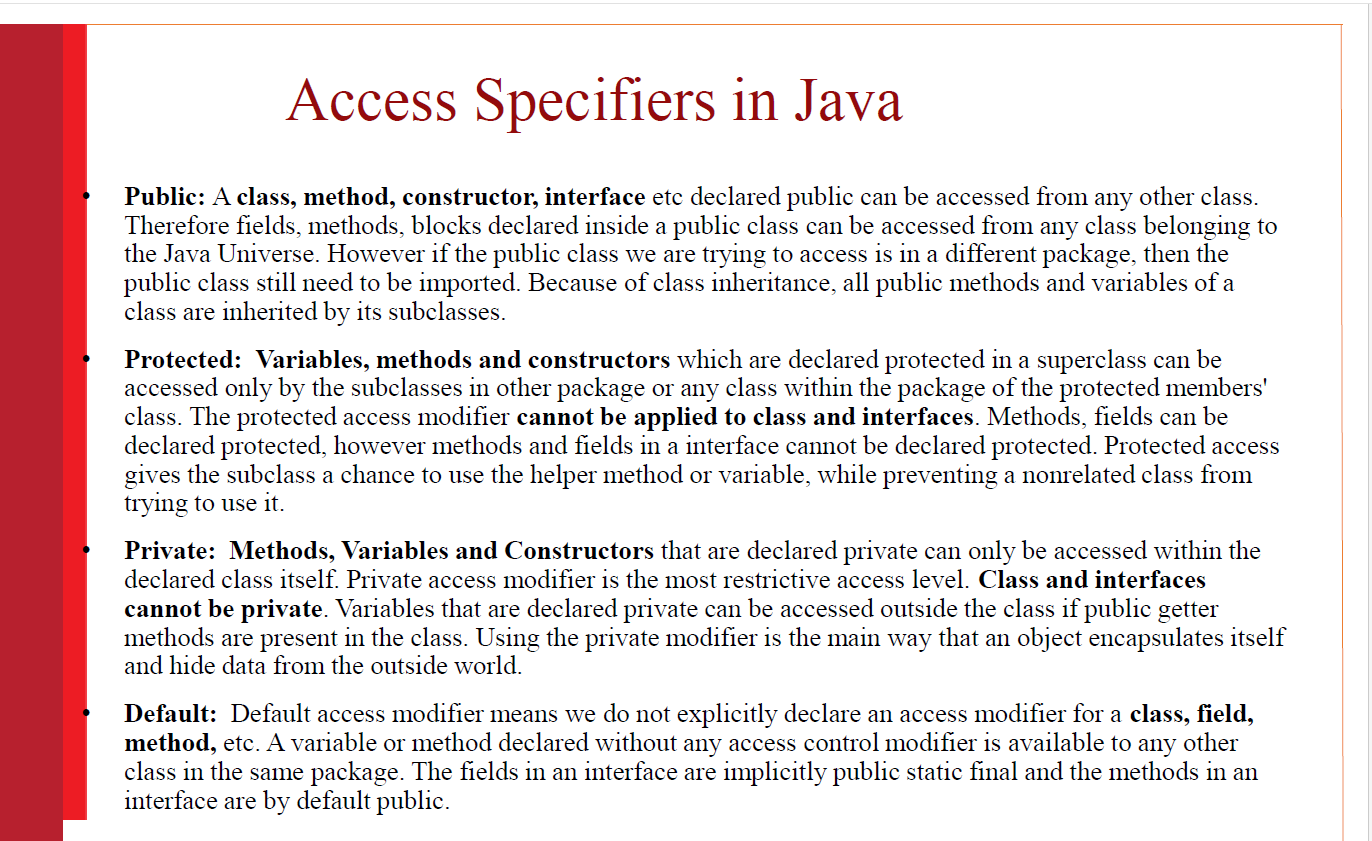


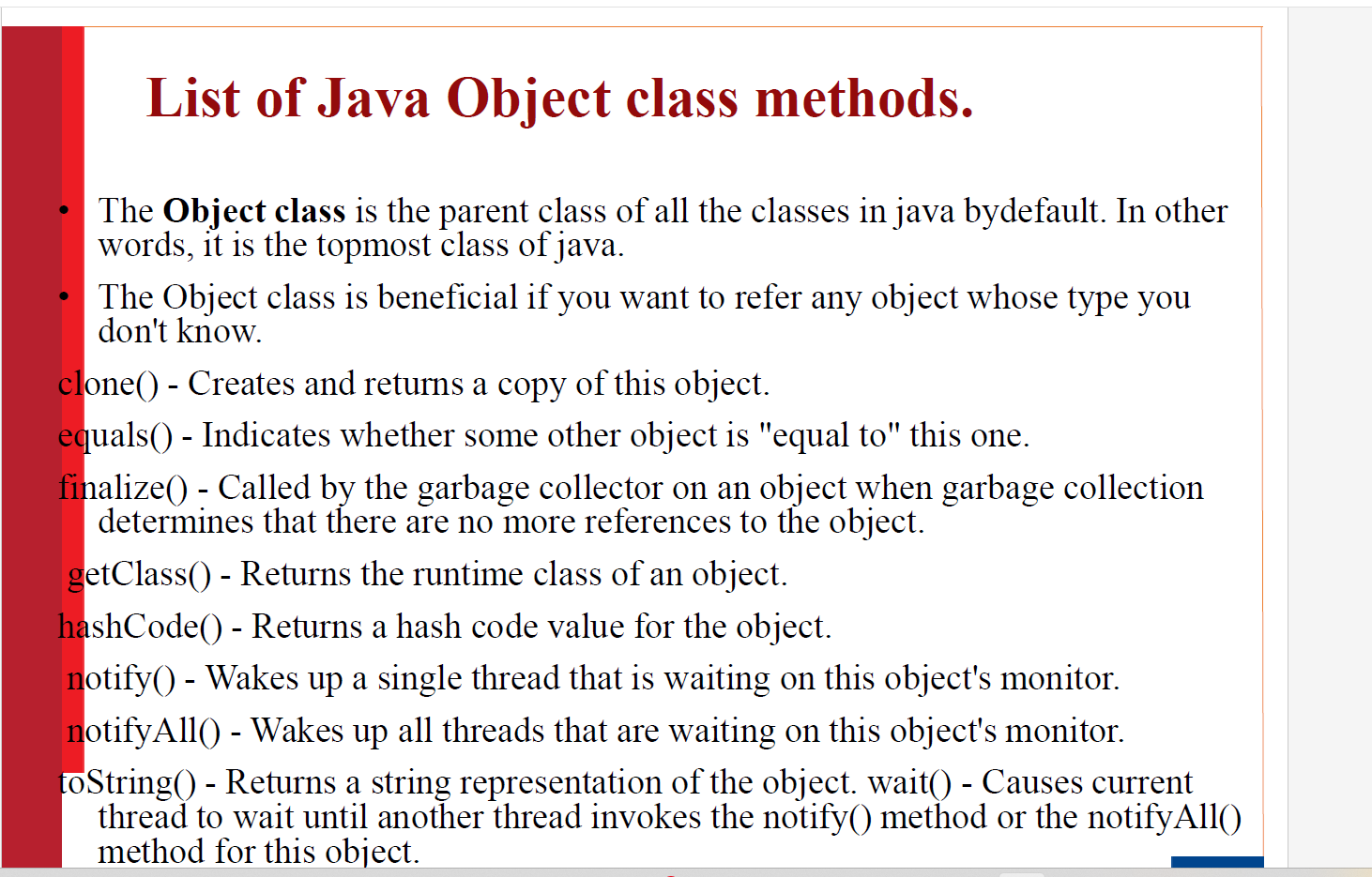


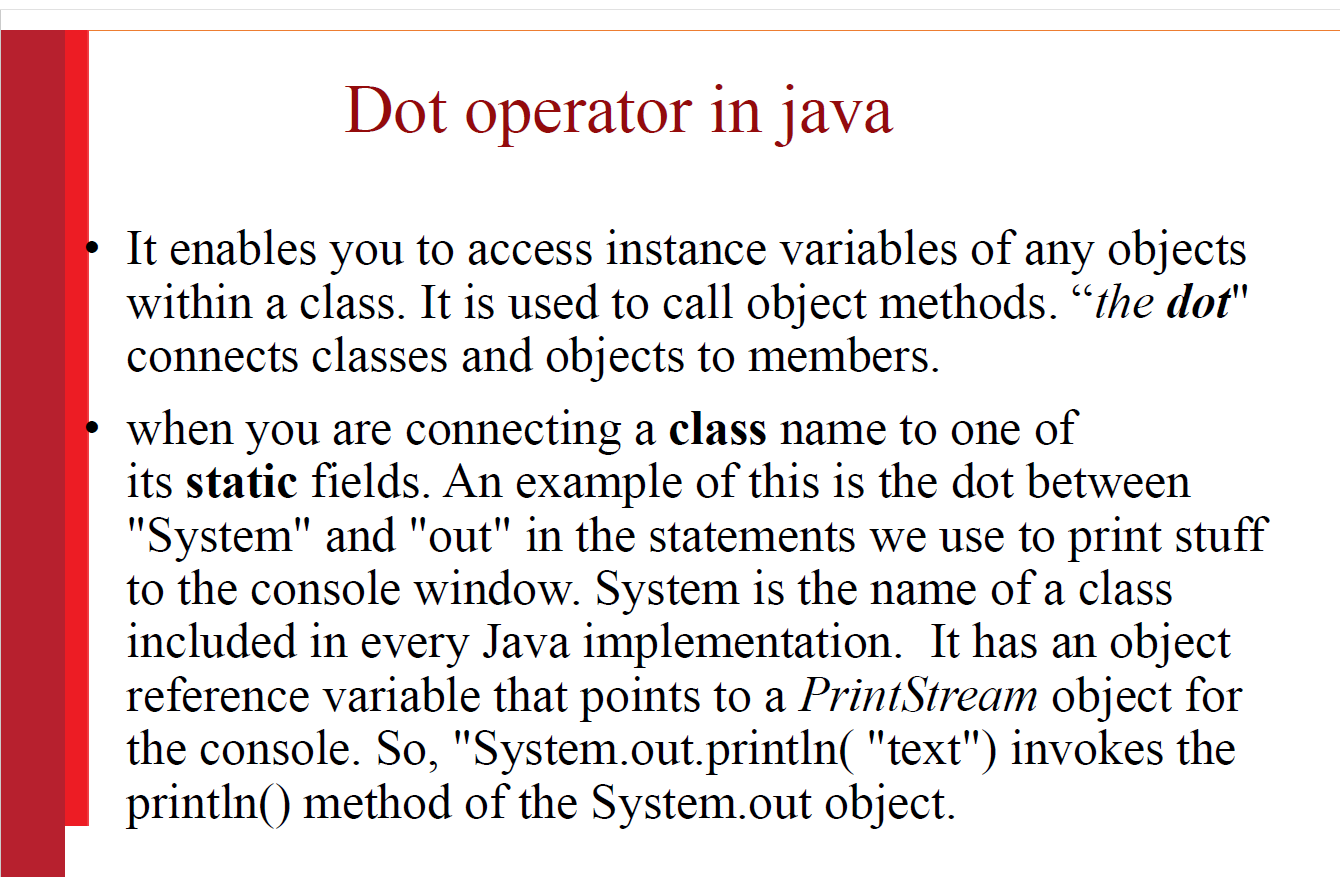


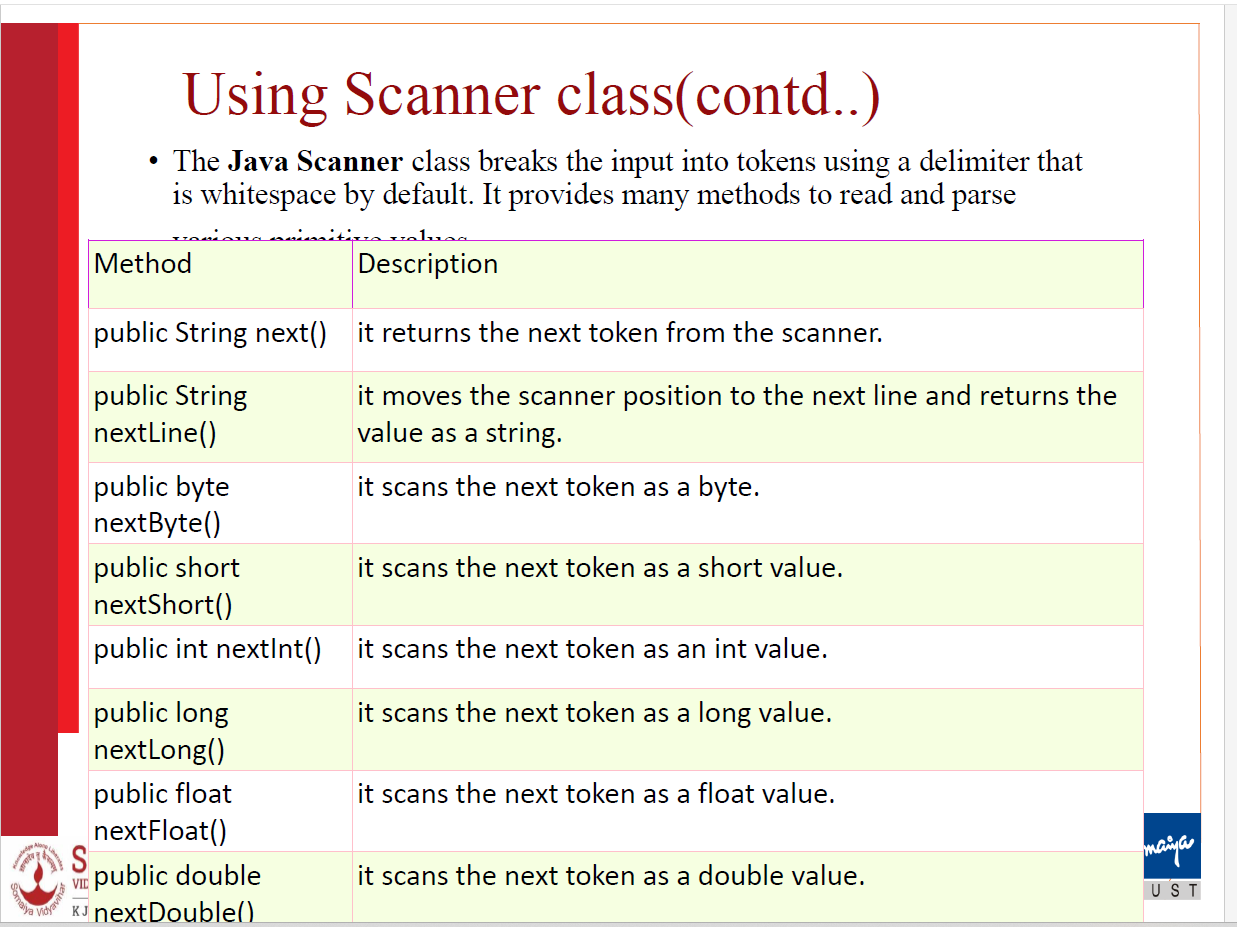


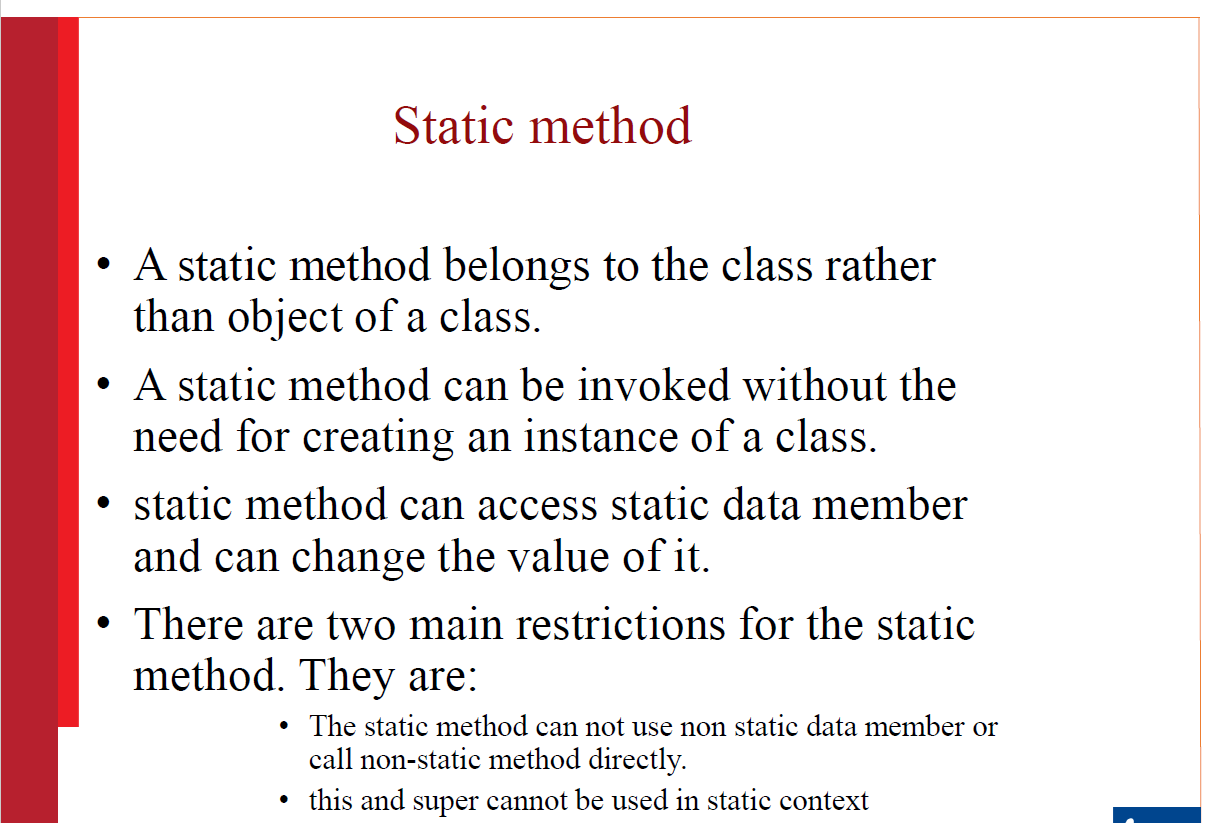












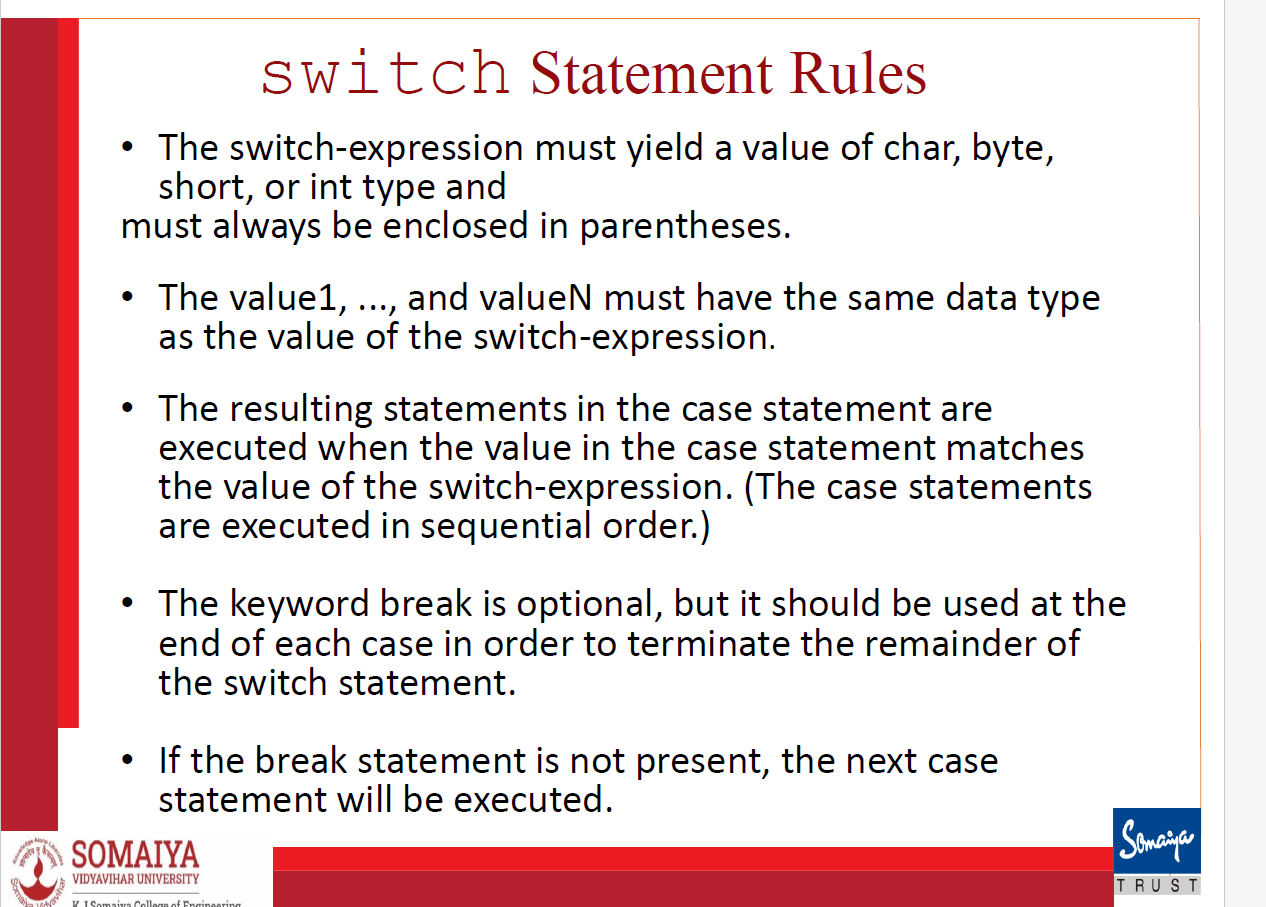
Static variable

•When a variable is declared with the keyword “static”, its called a “**class variable**”.

•All instances share the same copy of the variable. A class variable can be accessed directly with the class, without the need to create a instance. It makes your program **memory efficient.**

Static block

Is used to initialize the static data member. It is executed before main method at the time of classloading. So this is one of the way to **execute a program without main() method.**

****

