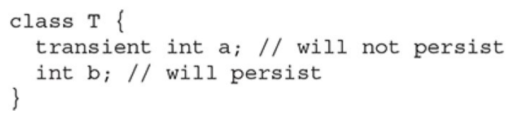
Transient Modifiers: in cases where an instance variable is specified as transient it will be excluded from Java serialization mechanism when an object is serialized to be stored to the output streams like files and databases (serialization is the process of converting an object from its in memory representation into a format where it can be easily stored transmitted and reconstructed later)

static variables are not part of the object's state and are not serialized.

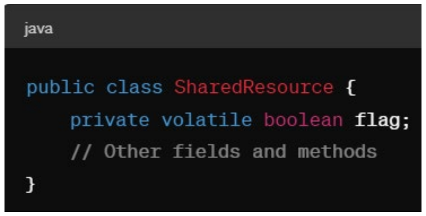
- It does not prevent the instance fields from being allocated memory or initialized when the object is created.
- When an object is serialized (converted into a byte stream for storage or transmission), only the values of non-transient fields are saved.

Serialization: Serialization is the process of converting an object from its in-memory representation into a format that can be easily stored, transmitted, or reconstructed later. In Java, serialization is achieved using the **java.io.Serializable interface**, which allows objects to be converted into a stream of bytes and then reconstructed back into an object

- Saving and restoring the state of objects in a persistent storage (e.g., files, databases).
- Transmitting objects between different applications or across a network.
- Caching and sharing objects in distributed systems
-



```java
class T {
  transient int a; // will not persist
  int b; // will persist
}
```



java

```java
public class SharedResource {
    private volatile boolean flag;
    // Other fields and methods
}
```

Why do we use transient modifier in healthcare applications

- Certain fields within a patient record might contain sensitive information, such as social security numbers, financial data, or personal identification numbers. To protect this sensitive data from being serialized and stored persistently, you could mark these fields as transient. This ensures that they are not included when the patient record objects are serialized and stored in the database.
- Some fields within a patient record might be derived from other fields or calculated dynamically. Examples include fields representing computed values like age, BMI (Body Mass Index), or risk scores. Since these fields can be recalculated or derived from other fields when needed, there may be no need to serialize them.
- By marking these fields as transient, you ensure that they are excluded from the serialization process, saving storage space and avoiding potential inconsistencies between the serialized data and the recalculated values.

Volatile modifiers: volatile is often used for flags or variables that are shared among multiple threads and whose values may change asynchronously.

- In multithreaded programming sometimes two or more threads share the same variable value, the volatile modifier is used to indicate that a variable's value may be modified by multiple threads that are executing concurrently.
- Volatile modifier specifies the compiler that the variables value may be modified by some other parts of the program,so it tells the compiler that it always must use the master copy of a volatile variable in order to avoid inconsistencies.

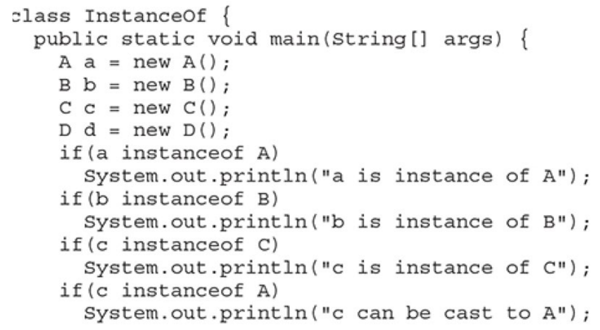
Why do we use volatile modifier in healthcare applications

- In multi-threaded healthcare applications, different threads may access and modify shared data concurrently. For example, in a real-time monitoring system, one thread might be responsible for updating patient vital signs while another thread is processing alarms or notifications.
- By marking shared variables as volatile, you ensure that changes made by one thread are immediately visible to other threads. This helps prevent data inconsistencies and ensures that all threads see the most up-to-date values of shared variables.
- For example, in a healthcare application where multiple threads are updating a shared counter representing the number of patients admitted to a hospital, marking the counter variable as volatile ensures that all threads see the latest count without the need for explicit synchronization.

instanceOf: This operator in Java is used to know the type of an object. The syntax of the instanceof operator is: **object instanceof Class.**

Here, object is the reference variable to be tested, and Class is the class or interface being tested against.

- An invalid casting can cause runtime errors we can identify the cast of an object using instance of and prevent the program from running into errors.

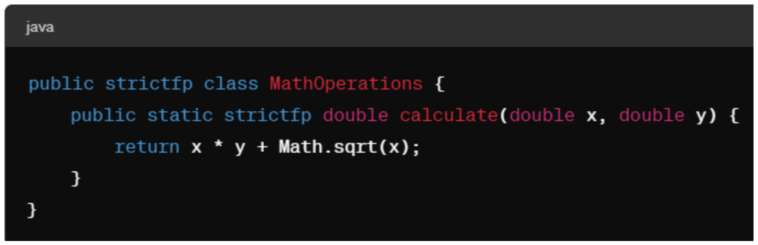


```
:lass InstanceOf {
  public static void main(String[] args) {
    A a = new A();
    B b = new B();
    C c = new C();
    D d = new D();
    if(a instanceof A)
      System.out.println("a is instance of A");
    if(b instanceof B)
      System.out.println("b is instance of B");
    if(c instanceof C)
      System.out.println("c is instance of C");
    if(c instanceof A)
      System.out.println("c can be cast to A");
```

- It's also used in type casting to ensure type safety before performing the casting operation.

Strictfp:

The strictfp keyword in Java is a modifier that can be applied to classes, interfaces, and methods. It stands for "strict floating-point" and affects the floating-point calculations performed within the scope of the entity to which it is applied.



```java
public strictfp class MathOperations {
    public static strictfp double calculate(double x, double y) {
        return x * y + Math.sqrt(x);
    }
}
```

In this example, the MathOperations class and the calculate method are declared with strictfp, ensuring that all floating-point calculations within the class/ method adhere to strict floating-point rules.

- Java program can run on any platform with JVM installed (platform independent). However different platforms may have different implementations of the JVM which could lead to variations in floating point arithmetic behavior.
- Strictfp specifies formats for representing floating-point numbers, rules for rounding, handling of special values (such as NaN and infinity), and requirements for arithmetic operations.