MongoDB vs MySQL

MySQL stores its data in tables and uses the structured query language (SQL) to access the data. MySQL uses schemas to define the database structure, requiring that all rows within a table have the same structure with values being represented by a specific data type

- MongoDB, data is stored in JSON-like documents that can have varied structures.
- MongoDB is schema-free, allowing you to create documents without having to define the structure of the document first
- In MongoDB, documents are able to have their own unique structure. New fields can be added at any time and contain any type of value. This type of functionality would require a relational database to be restructured.
- Using the MongoDB data model, you can represent hierarchical relationships, data arrays, and other complex structures in the database. In some cases, MongoDB performance is improved over MySQL because MongoDB does not use joins to connect data, improving performance

How to create index and how to fetch through indexes?

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book. An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

- **MySQL**: With MySQL, if an index is not defined, the database engine must scan the entire table to find all relevant rows.

  CREATE INDEX index_name **ON** table_name;
- **MongoDB**: In MongoDB, if an index is not found, every document within a collection must be scanned to select the documents that provide a match to

the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

To create an index in the Mongo Shell, use **db.collection.createIndex().**

Abstraction:

Abstract Class

Interface

Encapsulation:

Private variable

Getter and Setter Methods

Polymorphism:

Static – Method Overloading

Dynamic – Method Overriding

Inheritance:

Multi-level Inheritance

Association:

Association simply means the act of establishing a relationship between two unrelated classes. For example, when you declare two fields of different types (e.g. Car and Bicycle) within the same class and make them interact with each other, you have performed association.

- Two separate classes are associated through their objects.
- The two classes are unrelated, each can exist without the other one.
- Can be a one-to-one, one-to-many, many-to-one, or many-to-many relationship.

Aggregation

Aggregation is a narrower kind of association. It occurs when there's a one-way (HAS-A) relationship between the two classes you associate through their objects. For example, every Passenger has a Car but a Car doesn't necessarily have a Passenger. When you declare the Passenger class, you can create a field of the Car type that shows which car the passenger belongs to. Then, when you instantiate a new Passenger object, you can access the data stored in the related Car as well.

- One-directional association.
- Represents a HAS-A relationship between two classes.
- Only one class is dependent on the other.

Composition

Composition is a stricter form of aggregation. It occurs when the two classes you associate are mutually dependent on each other and can't exist without each other. For example, take a Car and an Engine class. A Car cannot run without an Engine, while an Engine also can't function without being built into a Car. This kind of relationship between objects is also called a PART-OF relationship.

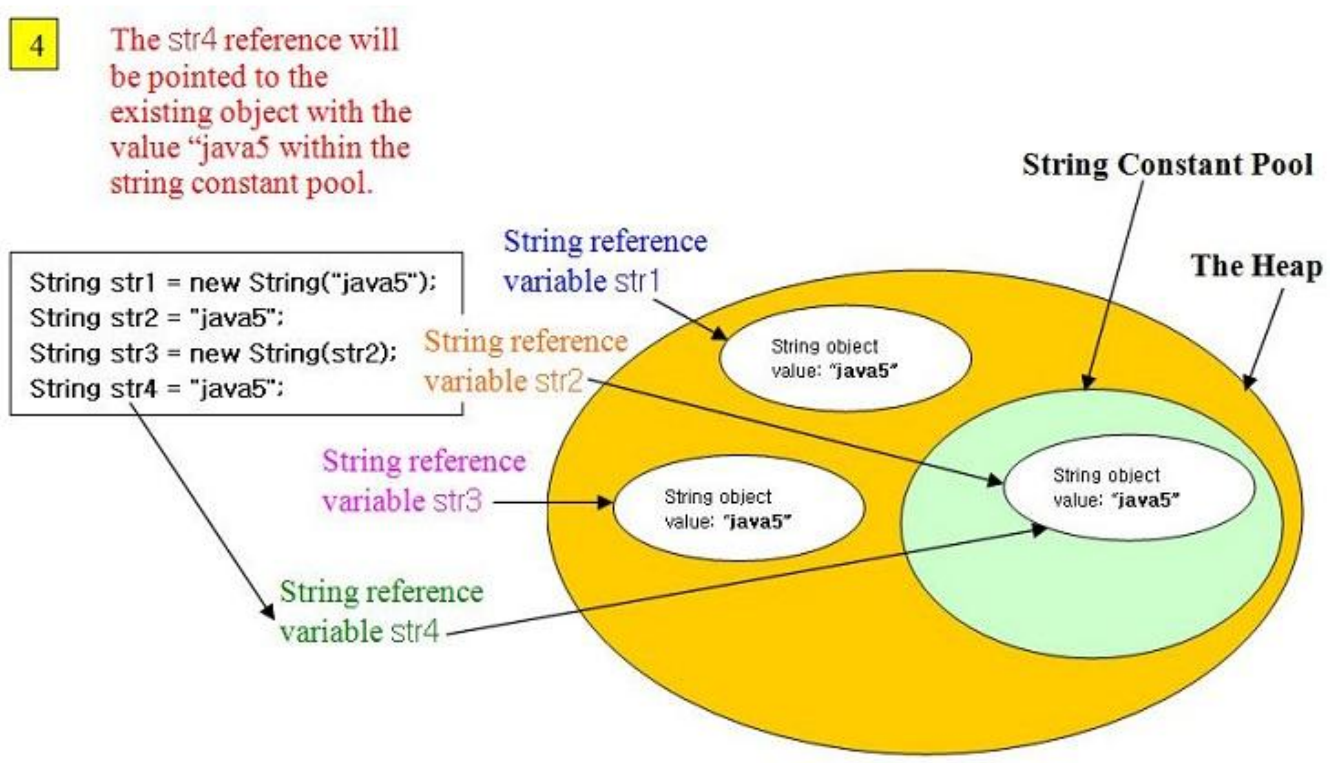
- A restricted form of aggregation
- Represents a PART-OF relationship between two classes
- Both classes are dependent on each other
- If one class ceases to exist, the other can't survive alone

Difference between String new () and String =""

A **string constant pool** is a separate place in the heap memory where the values of all the strings which are defined in the program are stored.

When we declare a string, an object of type String is created in the stack, while an instance with the value of the string is created in the heap.

On standard assignment of a value to a string variable, the variable is allocated stack, while the value is stored in the heap in the string constant pool


4
The str4 reference will be pointed to the existing object with the value "java5 within the string constant pool.
String str1 = new String("java5");
String str2 = "java5";
String str3 = new String(str2);
String str4 = "java5";
String Constant Pool
The Heap
String reference variable str1
String reference variable str2
String reference variable str3
String reference variable str4
String object value: "java5"
String object value: "java5"
String object value: "java5"

For str1 and str 3 two separate objects are created in "heap memory"

But, for str2 and str4 only one object is created and reference to the same.

Difference between == operator and equals() method

== checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects

If a class does not override the equals method, then by default it uses equals(Object o) method of the closest parent class that has overridden this method

---

Garbage Collection in Java:

Main objective of Garbage Collector is to free heap memory by destroying unreachable objects.

An object is said to be unreachable if it doesn't contain any reference to it.

Even though the programmer is not responsible to destroy useless objects but it is highly recommended to make an object unreachable(thus eligible for GC) if it is no longer required.

Once we made object eligible for garbage collection, it may not destroy immediately by the garbage collector. Whenever JVM runs the Garbage Collector program, then only the object will be destroyed. We can't predict when JVM shall run Garbage Collector.

We can also request JVM to run Garbage Collector. There are two ways to do it :

1. **Using *System.gc()* method** : System class contain static method *gc()* for requesting JVM to run Garbage Collector.
2. **Using *Runtime.getRuntime().gc()* method** : Runtime class allows the application to interface with the JVM in which the application is running

Finalization

- Just before destroying an object, Garbage Collector calls ***finalize****()* method on the object to perform cleanup activities. Once *finalize()* method completes, Garbage Collector destroys that object.
- *finalize()* method is present in **Object** class with following prototype.

  protected void finalize() throws Throwable

  Based on our requirement, we can override *finalize()* method for perform our cleanup activities like closing connection from database.

  Note :

  1. The *finalize()* method called by Garbage Collector not JVM. Although Garbage Collector is one of the module of JVM.
  2. Object class *finalize()* method has empty implementation, thus it is recommended to override *finalize()* method to dispose of system resources or to perform other cleanup.
  3. The *finalize()* method is never invoked more than once for any given object.
  4. If an uncaught exception is thrown by the *finalize()* method, the exception is ignored and finalization of that object terminates.

Can we overload static methods?

The answer is 'Yes'. We can have two or more static methods with the same name, but **differences in input parameters**. For example, consider the following Java program

Can we overload methods that differ only by static keyword?

We cannot overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is the same).

Can we Override static methods in java?

We can declare static methods with the same signature in the **subclass**, but it is not considered overriding as there won't be any run-time polymorphism. Hence the answer is '**No'**.

Where are Static variables stored in Java ?

When we create a static variable or method it is stored in the special area on heap: **PermGen(Permanent Generation)**, where it lays down with all the data applying to classes(non-instance data). Starting from Java 8 the PermGen became **-Metaspace**. The difference is that Metaspace is auto-growing space, while PermGen has a fixed Max size, and this space is shared among all of the instances. Plus the Metaspace is a part of a Native Memory and not JVM Memory.

Static methods are stored in Metaspace space of native heap as they are associated to the class in which they reside not to the objects of that class. But their local variables and the passed arguments are stored in the stack. Since static methods belong to the class - they can be called without creating an object of the class. The use-case of static methods is the same as with static variable.

What is cloning in java and how it is done ?

Object cloning refers to the creation of an exact copy of an object. It creates a new instance of the class of the current object and initializes all its fields with exactly the contents of the corresponding fields of this object.

The class must also implement **clone()** of **Cloneable interface** whose object clone we want to create otherwise it will throw CloneNotSupportedException when clone method is called on that class's object.

Advantages of Cloning in Java

- Helps in reducing the lines of code.
- The most effective and efficient way of copying objects.
- Also, the clone() is considered to be the fastest method to copy an array.

Shallow Cloning

In Java, when the cloning process is done by invoking the clone() method it is called Shallow Cloning. <u>In other words, if you change the value of the cloned objects then it will be reflected in the original as well. Thus, shallow cloning is dependent on the original object.</u>

Deep Cloning in Java

The cloned object independent of the original object and any changes made in any of the object won't be reflected on the other.

Serialization and DeSerialization

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

To make a Java object serializable we implement the **java.io.Serializable** interface.

Advantages of Serialization

1. To save/persist state of an object.
2. To travel an object across a network.

Points to remember

1. If a parent class has implemented Serializable interface then child class doesn't need to implement it but vice-versa is not true.
2. Only non-static data members are saved via Serialization process.
3. **Static** data members and **transient** data members are not saved via Serialization process. So, if you don't want to save value of a non-static data member then make it transient.
4. Constructor of object is never called when an object is deserialized.
5. Associated objects must be implementing Serializable interface.

SerialVersionUID

The Serialization runtime associates a version number with each Serializable class called a SerialVersionUID, which is used during Deserialization to verify that sender and reciever of a serialized object have loaded classes for that object which are compatible with respect to serialization.

If the reciever has loaded a class for the object that has different UID than that of corresponding sender's class, the Deserialization will result in an **InvalidClassException** . A Serializable class can declare its own UID explicitly by declaring a field name. It must be **static**, **final** and of type **long**.

```
ANY-ACCESS-MODIFIER static final long serialVersionUID= 42L;
```

It is also recommended to use private modifier for UID since it is not useful as inherited member.

If a serializable class doesn't explicitly declare a serialVersionUID, then the serialization runtime will calculate a default one for that class based on various aspects of class, as described in Java Object Serialization Specification. However it is strongly recommended that all serializable classes explicitly declare serialVersionUID value

The **ObjectOutputStream** class contains **writeObject()** method for serializing an Object

The **ObjectInputStream** class contains **readObject()** method for deserializing an object.

```
class Demo implements java.io.Serializable
{
    public int a;
    public String b;

    // Default constructor
    public Demo(int a, String b)
    {
        this.a = a;
        this.b = b;
    }
}



class Test
{
    public static void main(String[] args)
    {
        Demo object = new Demo(1, "geeksforgeeks");
// Serialization
       try
        {
//Saving of object in a file
            FileOutputStream file = new FileOutputStream(“texting.ser”);
            ObjectOutputStream out = new ObjectOutputStream(file);

// Method for serialization of object
            out.writeObject(object);

            out.close();
            file.close();

            System.out.println("Object has been serialized");

        }

        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }
      Demo object1 = null;

// Deserialization
        try
        {
// Reading the object from a file
            FileInputStream file = new FileInputStream(“texting.ser”);
            ObjectInputStream in = new ObjectInputStream(file);

// Method for deserialization of object
            object1 = (Demo)in.readObject();
            in.close();
            file.close();

            System.out.println("Object has been deserialized ");
            System.out.println("a = " + object1.a);
            System.out.println("b = " + object1.b);
        }

        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }
```

In case of **transient variables**:- A variable defined with transient keyword is not serialized during serialization process.This variable will be initialized with default value during deserialization. (e.g: for objects it is null, for int it is 0).
In case of **static Variables:-** A variable defined with static keyword is not serialized during serialization process.This variable will be loaded with current value defined in the class during deserialization.

Difference between >> operator and >>> operator ?

Both >> and >>> are used to shift the bits towards the right. The difference is that the >> preserve the sign bit while the operator >>> does not preserve the sign bit. To preserve the sign bit, you need to add 0 in the MSB