DeepSeek Chat

```
//Lecture 13
    //When entering values 10.1and 10.2
    /*
    double a, b, c;
    a = Double.parseDouble (a);
    b = Double.parseDouble (b);
    c = a+b;
    System.out.println (c);

    //Double, Integer, and Character are Wrapper Classes.
    //More on Wrapper Classes.
    //Run this program using command on CLI
    //passing parameters 10.1and 20.4
    //like this java Addition 10.120.4 into args[0] and args[1]

    //Parse double will also give the numberformatexception if we put in some values it can't handle
    // like java Addition 10a 20b
    //Java has 8 wrapper classes for 8 dataTypes that are primitives . VVIMP will be useful in last topic

    //Wrapper Classes in Java are those special classes whicha re available in the package java.lang and
    are very useful for programmers .

    These classes match with the names of their corresponding dataTypes.
    for int there is class Integer.
    similarly for every other primitive dataType we have a Wrapper Class present .
    Q.What is the use of Wrapper class?
    A. 2 Uses in Java for Wrapper Classes.
    Wrapper Classes in Java are used for 2 purposes :
```

Wrapper Classes in Java are used for 2 purposes .

1.To convert String representation of a primitive value to it's actual representation .

ex: "25" String converted to 25 int done by the Wrapper class Integer with the help of the method parseInt ().

There are parse methods like parseInt () for each of the Wrapper classes except for char. See chart

q. Does char have a wrapper class ? Does it have a parse method ?

a. yes it has a wrapper class , but it does not have a parse method .

2. Wrapper Classes can be used to convert variables into objects .

//a variable called a of type int is declared and initialized as 10.

int a = 10;

a is converted into an object , the primitve value a is converted to the object obj which is of the wrapper class Integer .

Integer obj = a;

//This will be useful in chapter collections .

All common operations on data can be done using collections without making methods for it ourself .

in java int is different , and Integer Wrapper class are different , this topic is the last topic collections and we see another use

for wrapper classes . VVIMP for Java and in Java interviews called Collections .

This is because collections allow for us to perform standard operations on a group of

data like arrays by just calling simple methods . So for searching sorting , deleting adding data

in any array or dataType we can call methods instead of writing our own logic .

Integer is a top level class meaning it has no classes that hold this class inside it, it is not a child

Q. can you make an outer class static ?

A. no you can only make an inner class static never an outer class

ex:

```
static class A{
//code nope doesn't run gives error modifier static not allowed here
//outer classes can never be static.
}
// works
class A{

static class B{
   }
}
//works and a class within another is a nested class concept.
class A{

class B{
   }
}
```

Q. where can we use static in java?

1.static data

2. static method

3. static initializer block

4. only inner classes as static in java

Topic

A lot of programs in java require String methods

to use them we will have to make String objects to use the methods

Using methods of the String class. (Its not a wrapper class nor dataType, its a class)

String has very useful methods for performing basic operations on strings,

maximum methods are nonstatic in this class, some are instance methods,

so we will have to make objects.

Java

```
class Student{
int rol;
char grade;
float per;

get();

show();

}
//This is not an object, this is an object reference S variable.
//we need the new keyword to make an object of Student class.
//The objects in Java are in the Heap area.
//Because anything made in runtime, they are stored in heap.
//Objects in java are dynamic, they are made in runtime.
//need to say S = new Student();
Student S;
S = new Student();
```

S recieves this object, object reference is like a restricted pointer.

object reference only holds the address of the object, and the object lives in the heap.

object references live in the heap area.

S is not an object, S is an object reference.

We use . to access the methods of the object and the object reference has the .

S gets the address.

You can also write Student S = new Student();

combined

String is a predefined class so we just make an object of that class.

Using "String" class

==========

```
classTest{
psvm{
```

String s1; //object reference s1

s1 = new String("Bhopal"); // to make a new object it needs new keyword and write the String and constructor ()..

// THis makes the object inside the heap and allocates space for it there

String s2 = new String("Bhopal"); // Same happens here but their locations in the heap are different , theyre addresses .

System.out.println(s1 == s2); //Gives false, because these are object references holding address of objects that say same thing "Bhopal","Bhopal"

}

1. To initialize String objects we pass string valyes within parenthesis while creating the object /

2. Whenever we display a String object reference we do not get address displayed rather we get the contents

of the String object pointed by that reference .

3. But if we compare 2 String references then Java compares addresses and returns trur or false

depending on whether the references are pointing to the same object or not.

}

//references in java occupy 8 bytes. converting 64 bits to bytes its 8 bytes .

*/

Make detailed and organized notes on these concepts based on the notes I have taken, make sure they are correct and have explanations .

Detailed and Organized Notes on Java Concepts

1. **Wrapper Classes in Java**

- **Definition**: Wrapper classes are special classes in Java that are part of the `java.lang` package. They are used to convert primitive data types into objects and vice versa.
- **Purpose**:
  1. **Conversion**: Wrapper classes are used to convert the string representation of a primitive value into its actual primitive type. For example, converting the string `"25"` to the integer `25` using the `Integer.parseInt()` method.
  2. **Object Conversion**: Wrapper classes can also be used to convert primitive variables into objects. For example, converting an `int` variable into an `Integer` object.
- **Examples of Wrapper Classes**:
  - `Integer` for `int`
  - `Double` for `double`
  - `Character` for `char`
  - `Boolean` for `boolean`
  - etc.
- **Important Methods**:
  - `Integer.parseInt(String s)`: Converts a string to an integer.
  - `Double.parseDouble(String s)`: Converts a string to a double.
  - `Character.valueOf(char c)`: Converts a char to a Character object.
- **Exception Handling**: Methods like `parseInt()` and `parseDouble()` can throw a `NumberFormatException` if the string cannot be parsed into the respective primitive type (e.g., `java Addition 10a 20b`).

2. **Static Keyword in Java**

- **Usage**:
  1. **Static Data**: Variables declared as `static` belong to the class rather than any specific instance.
  2. **Static Methods**: Methods declared as `static` can be called without creating an instance of the class.
  3. **Static Initializer Block**: A block of code that runs when the class is first loaded into memory.

4. **Static Inner Classes**: Only inner classes can be declared as `static`. Outer classes cannot be static.

- **Example**:

```java
class A {
    static class B {
        // Static inner class
    }
}
```

- **Key Points**:
  - Outer classes cannot be static.
  - Static members are shared across all instances of the class.

3. **String Class in Java**

- **Definition**: The `String` class in Java is a predefined class used to represent a sequence of characters. It is not a primitive data type but a class.
- **Creating String Objects**:
  - **Using `new` Keyword**:

```java
String s1 = new String("Bhopal");
String s2 = new String("Bhopal");
```

    - Here, `s1` and `s2` are two different objects with different memory addresses, even though they contain the same string value.
  - **String Literal**:

```java
String s3 = "Bhopal";
String s4 = "Bhopal";
```

    - Here, `s3` and `s4` point to the same memory location in the string pool.
- **String Comparison**:
  - **Using `==`**: Compares the memory addresses of the string objects.

```java
System.out.println(s1 == s2); // false
System.out.println(s3 == s4); // true
```

```
System.out.println(s3 == s4); // true
```

- **Using** `.equals()`: Compares the actual content of the strings.

```java
System.out.println(s1.equals(s2)); // true
```

- **Key Points**:
  - String objects created with the `new` keyword are stored in the heap memory.
  - String literals are stored in the string pool, which is a part of the heap memory.
  - String references in Java occupy 8 bytes (64 bits).

4. **Objects and Object References in Java**

- **Definition**:
  - **Object**: An instance of a class that is created at runtime and stored in the heap memory.
  - **Object Reference**: A variable that holds the memory address of an object. It is not the object itself but a pointer to the object.
- **Creating Objects**:

```java
Student S; // Object reference
S = new Student(); // Object creation
```

  - Here, `S` is an object reference that points to the `Student` object created in the heap memory.
- **Accessing Object Members**:
  - Use the dot (`.`) operator to access methods and variables of an object.

```java
S.get(); // Accessing the get() method of the Student object
```

- **Key Points**:
  - Objects are dynamic and created at runtime.
  - Object references are like restricted pointers that hold the address of the object.

5. **Collections in Java**

- **Definition**: Collections in Java are used to store, retrieve, and manipulate groups of data. They provide standard operations like searching, sorting, adding, and deleting data.
- **Wrapper Classes in Collections**:
  - Collections in Java work with objects, not primitive types. Therefore, wrapper classes are used to convert primitive types into objects.
  - Example:

```java
int a = 10;
Integer obj = a; // Converting primitive to object
```

- **Benefits**:
  - Collections provide built-in methods for common operations, reducing the need to write custom logic.
  - They are essential for handling groups of data efficiently.

6. **Common String Methods**

- **Non-static Methods**: Most methods in the `String` class are non-static, meaning they require an object to be called.
- **Examples**:
  - `length()`: Returns the length of the string.
  - `charAt(int index)`: Returns the character at the specified index.
  - `substring(int beginIndex)`: Returns a substring from the specified index to the end.
  - `equals(Object obj)`: Compares the content of two strings.
- **Example Usage**:

```java
String s = new String("Hello");
System.out.println(s.length()); // 5
System.out.println(s.charAt(1)); // 'e'
System.out.println(s.substring(2)); // "llo"
```

7. **Heap Memory in Java**

- **Definition**: Heap memory is the runtime data area where objects are stored. It is shared among all threads in a Java application.
- **Key Points**:
  - Objects created with the `new` keyword are stored in the heap.
  - Object references are stored in the stack, but they point to objects in the heap.
  - Heap memory is managed by the Java Virtual Machine (JVM).

Summary

- **Wrapper Classes**: Used for converting primitives to objects and vice versa. Essential for collections and parsing strings.
- **Static Keyword**: Used for class-level variables, methods, and inner classes. Outer classes cannot be static.
- **String Class**: Represents a sequence of characters. Strings can be created using literals or the `new` keyword. String comparison can be done using `==` (address comparison) or `.equals()` (content comparison).
- **Objects and References**: Objects are instances of classes stored in the heap, while object references are variables that hold the address of these objects.
- **Collections**: Used for handling groups of data. Wrapper classes are essential for working with collections.
- **Heap Memory**: The area where objects are stored at runtime. Object references point to objects in the heap.

These notes should help you understand the key concepts discussed in the lecture and provide a solid foundation for further study in Java programming.

How many wrapper classes can be initialized without using "new" in Java

In Java, **wrapper classes** can be initialized without using the `new` keyword through a feature called **autoboxing**. Autoboxing allows the Java compiler to automatically convert primitive types into their corresponding wrapper class

objects .

Wrapper Classes in Java

There are **8 wrapper classes** in Java, corresponding to the 8 primitive data types:

1. `Integer` (for `int`)
2. `Double` (for `double`)
3. `Float` (for `float`)
4. `Long` (for `long`)
5. `Short` (for `short`)
6. `Byte` (for `byte`)
7. `Character` (for `char`)
8. `Boolean` (for `boolean`)

Initialization Without `new`

All **8 wrapper classes** can be initialized without using the `new` keyword. This is because of **autoboxing**, which was introduced in Java 5. Autoboxing allows you to directly assign a primitive value to a wrapper class object, and the Java compiler automatically converts the primitive into its corresponding wrapper class object.

Examples of Initialization Without `new`:

```java
Integer i = 10;              // Autoboxing: int to Integer
Double d = 10.5;             // Autoboxing: double to Double
Float f = 10.5f;             // Autoboxing: float to Float
Long l = 100L;               // Autoboxing: long to Long
Short s = 5;                 // Autoboxing: short to Short
Byte b = 1;                  // Autoboxing: byte to Byte
Character c = 'A';           // Autoboxing: char to Character
Boolean bool = true;         // Autoboxing: boolean to Boolean
```

Key Points:

1. **Autoboxing**: Automatically converts primitive types to their corresponding wrapper class objects.
2. **No `new` Keyword Needed**: You can directly assign primitive values to wrapper

class objects .

3. **All 8 Wrapper Classes Support Autoboxing**: This means all 8 wrapper classes can be initialized without explicitly using the `new` keyword .

Why Avoid `new` for Wrapper Classes ?

Using `new` to create wrapper class objects is **not recommended** because :

- It creates a new object in the heap every time , which is less efficient .
- Autoboxing is more concise and easier to read .
- Wrapper classes like `Integer` , `Character` , etc ., have a **cache** for commonly used values (e.g., `Integer` caches values from -128 to 127). Using `new` bypasses this cache , leading to unnecessary object creation .

Example with `new` (Not Recommended):

```java
Integer i = new Integer(10);   // Creates a new object , bypassing the cache
Double d = new Double(10.5);   // Creates a new object
```

Example Without `new` (Recommended):

```java
Integer i = 10;   // Uses autoboxing and may reuse cached values
Double d = 10.5; // Uses autoboxing
```

Conclusion :

All **8 wrapper classes** in Java can be initialized without using the `new` keyword , thanks to **autoboxing** . This feature makes the code cleaner , more efficient , and easier to read . Always prefer autoboxing over explicit `new` for wrapper class initialization .

Also , from my notes , with answers and their explanations derived from them , I want you to prepare questions and answers on this topic based on the questions in my notes right now , and do it well and proper , don't at all veer off topic , stay on point , get it done .

Here are **questions and answers** based on the notes you provided , strictly adhering to the topics and explanations in your notes . These questions are designed to test understanding of the concepts discussed in your notes .

Questions and Answers on Wrapper Classes, Static Keyword, String Class, and Objects

---

1. Wrapper Classes

Q1. What are wrapper classes in Java?

A1. Wrapper classes in Java are special classes in the `java.lang` package that allow primitive data types to be used as objects . They "wrap" primitive types into objects , enabling them to be used in contexts where objects are required , such as collections .

Q2. What are the two main uses of wrapper classes in Java?

A2.

1. **Conversion**: Wrapper classes convert the string representation of a primitive value into its actual primitive type (e.g., `Integer.parseInt("25")` converts `"25"` to `25`).
2. **Object Conversion**: Wrapper classes convert primitive variables into objects (e.g., `int a = 10; Integer obj = a;`).

Q3. Which wrapper class does not have a `parse` method?

A3. The `Character` wrapper class does not have a `parse` method . It uses `valueOf()` instead (e.g., `Character.valueOf('A')`).

Q4. What exception is thrown if a string cannot be parsed into a primitive type using wrapper class methods?

A4. A `NumberFormatException` is thrown if the string cannot be parsed into a primitive type (e.g., `Integer.parseInt("10a")`).

Q5. Can all wrapper classes be initialized without using the `new` keyword?

Q5. Can all wrapper classes be initialized without using the `new` keyword?

A5. Yes, all 8 wrapper classes (`Integer`, `Double`, `Float`, `Long`, `Short`, `Byte`, `Character`, `Boolean`) can be initialized without using the `new` keyword due to **autoboxing**.

---

2. Static Keyword

Q6. Can an outer class be declared as `static` in Java?

A6. No, an outer class cannot be declared as `static`. Only inner classes can be declared as `static`.

Q7. Where can the `static` keyword be used in Java?

A7. The `static` keyword can be used in:

1. **Static Data**: Class-level variables.
2. **Static Methods**: Methods that belong to the class rather than an instance.
3. **Static Initializer Blocks**: Code blocks that run when the class is loaded.
4. **Static Inner Classes**: Inner classes declared as `static`.

Q8. What is the difference between a static inner class and a non-static inner class?

A8.

- A **static inner class** does not require an instance of the outer class to be instantiated.
- A **non-static inner class** requires an instance of the outer class to be instantiated.

---

3. String Class

Q9. How are String objects created in Java?

A9. String objects can be created in two ways:

1. Using the `new` keyword:

```java
String s1 = new String("Bhopal");
```

2. Using string literals:

```java
String s2 = "Bhopal";
```

Q10. What is the difference between `==` and `.equals()` when comparing strings?

A10.

- `==` compares the memory addresses of the string objects.
- `.equals()` compares the actual content of the strings.

Example:

```java
String s1 = new String("Bhopal");
String s2 = new String("Bhopal");
System.out.println(s1 == s2);          // false (different addresses)
System.out.println(s1.equals(s2)); // true (same content)
```

Q11. Where are String objects stored in memory?

A11.

- String objects created with the `new` keyword are stored in the **heap memory**.
- String literals are stored in the **string pool**, which is a part of the heap memory.

Q12. What is the size of a String object reference in Java?

A12. A String object reference occupies **8 bytes** (64 bits) in Java.

---

4. Objects and Object References

Q13. What is the difference between an object and an object reference in Java?

A13.

- An **object** is an instance of a class created at runtime and stored in the heap memory.
- An **object reference** is a variable that holds the memory address of the object.

It is not the object itself but a pointer to the object.

Q14. How do you create an object in Java?

A14. Objects are created using the `new` keyword. For example:

```java
Student S;              // Object reference
S = new Student();      // Object creation
```

Q15. What does the dot (`.`) operator do in Java?

A15. The dot (`.`) operator is used to access the methods and variables of an object through its reference. For example:

```java
S.get();  // Accessing the get() method of the Student object
```

---

5. Collections

Q16. Why are wrapper classes important in collections?

A16. Collections in Java work with objects, not primitive types. Wrapper classes are used to convert primitive types into objects so they can be stored in collections.

Q17. What is autoboxing in Java?

A17. Autoboxing is the automatic conversion of primitive types into their corresponding wrapper class objects by the Java compiler. For example:

```java
int a = 10;
Integer obj = a;  // Autoboxing: int to Integer
```

Q18. What is the benefit of using collections in Java?

A18. Collections provide built-in methods for common operations like searching, sorting, adding, and deleting data, reducing the need to write custom logic.

---

6. Heap Memory

Q19. What is heap memory in Java?

A19. Heap memory is the runtime data area where objects are stored. It is shared among all threads in a Java application.

Q20. Where are objects stored in Java?

A20. Objects are stored in the **heap memory**, while object references are stored in the **stack memory**.

Summary

These questions and answers are designed to test your understanding of the key concepts in your notes, including wrapper classes, the `static` keyword, the `String` class, objects and references, collections, and heap memory. Each answer is derived directly from the explanations in your notes, ensuring accuracy and relevance.