Wrapper Classes

In Java, wrapper classes are part of the Java Standard Library and provide a mechanism to "wrap" primitive data types in an object so that they can be used in situations where only objects are usable. Wrapper classes convert primitive data types (like `int`, `char`, `double`) into their corresponding object representations.

List of Wrapper Classes:

1. **Byte**: Wraps the `byte` primitive data type.
2. **Short**: Wraps the `short` primitive data type.
3. **Integer**: Wraps the `int` primitive data type.
4. **Long**: Wraps the `long` primitive data type.
5. **Float**: Wraps the `float` primitive data type.
6. **Double**: Wraps the `double` primitive data type.
7. **Character**: Wraps the `char` primitive data type.
8. **Boolean**: Wraps the `boolean` primitive data type.

Example 1:

```java
public class Main {
    public static void main(String[] args) {
        int num = 100;
        // Autoboxing: Converting int to Integer
        Integer num2 = num;

        // explicit boxing
        Integer num3 = Integer.valueOf(num);

        // Printing object
        System.out.println(num3.getClass().getName());
        // Outputs: java.lang.Integer
        System.out.println(num3.doubleValue());
        // Outputs: 100.0

        // Unboxing: Converting Integer to int
        int num2 = num3.intValue(); // or simply int num2 = num3;

        // Printing primitive data types
        System.out.println(num2);
    }
}
```

1. **Explicit Boxing:** **`Integer.valueOf(num)`**
    - This is an explicit call to the `valueOf` method of the `Integer` class. It's a clear indication in the code that boxing is taking place.
    - `Integer.valueOf(num)` checks if the value of `num` is within the range of -128 to 127(Integer Cache range). If so, it returns a reference to a pre-existing object from the Integer cache, which is a memory-saving feature.
2. **Autoboxing:** **`Integer num2 = num;`**
    - This is an example of autoboxing, where the Java compiler automatically converts the primitive `int` to an `Integer` object.
    - Internally, autoboxing does the same thing as `Integer.valueOf(num)`. It uses the `valueOf` method, so it also benefits from the Integer cache.

Example 2: Parsing Strings to Numbers

Wrapper classes provide utility methods to convert strings into numbers.

```java
public class Main {
    public static void main(String[] args) {
        String numberStr = "12345";
        int number = Integer.parseInt(numberStr);  // Convert String to int - parseInt
        System.out.println(number + 10);  // Outputs: 12355 which is an primitive integ

        Integer num3 = Integer.valueOf(number); // the valueOf method returns an Intege
        int num4 = Integer.valueOf(number); // the valueOf method returns an Integer ob
        System.out.println(num3 + 10); // Outputs: 12355 which is an object of Integer
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        int primitiveInt = 42;
        Integer wrappedInt = Integer.valueOf(primitiveInt);

        printType(primitiveInt);  // Outputs: The type of the variable is: int
        printType(wrappedInt);    // Outputs: The type of the variable is: Integer
    }

    public static void printType(int value) {
        System.out.println("The type of the variable is: int");
    }

    public static void printType(Integer value) {
        System.out.println("The type of the variable is: Integer");
    }

    // ... add more overloaded methods for other types if needed
}
```

Example 3: Checking Numeric Properties

The `Character` wrapper class provides methods to check properties of characters.

```java
public class Main {
    public static void main(String[] args) {
        char ch = '7';
        if (Character.isDigit(ch)) {
            System.out.println(ch + " is a digit.");  // Outputs: 7 is a digit.
        }
    }
}
```

Key Points about Wrapper Classes

1. **Object Representation**: Wrapper classes convert primitive data types (like `int`, `char`, `double`) into their corresponding object representations.
2. **Class Hierarchy**: All wrapper classes are subclasses of the `Object` class.
3. **Immutable**: Objects of wrapper classes are immutable, meaning their state cannot be changed once they are created.
4. **Autoboxing and Unboxing**: Java provides automatic conversion between primitive types and their corresponding wrapper objects. This feature was introduced in Java 5.
    - **Autoboxing**: Automatic conversion of primitive types to their corresponding wrapper class objects.
    - **Unboxing**: Automatic conversion of wrapper class objects back to their corresponding primitive types.

Why Use Wrapper Classes?

1. **Generics**: Java's generics do not support primitive types. So, if you want to use a primitive type with a generic class or method, you'll need to use its corresponding wrapper class.
2. **Collections Framework**: The Java Collections Framework (like `ArrayList`, `HashMap`, etc.) stores objects and not primitives. Wrapper classes enable storing primitive values in collections.
3. **Null Handling**: Primitive types cannot be `null`, but their wrapper objects can be. This can be useful in scenarios where you might want to represent the absence of a value.
4. **Additional Methods**: Wrapper classes come with a variety of utility methods. For instance, the `Integer` class provides methods like `parseInt`, `valueOf`, and others that can be very useful for operations related to integers.

A Naive Example of Finding the type of a variable

```java
public class Main {
    public static void main(String[] args) {
        int primitiveInt = 42;
        Integer wrappedInt = Integer.valueOf(42);

        System.out.println(typeOf(primitiveInt)); // Outputs: The type of the variable
        System.out.println(typeOf(wrappedInt));  // Outputs: The type of the variable i
    }

    // using method overloading to find the type of a variable
    public static String typeOf(int value) {
        return "int";
    }

    public static String typeOf(Integer value) {
        return "Integer";
    }

    public static String typeOf(double value) {
        return "double";
    }

    public static String typeOf(Double value) {
        return "Double";
    }

    public static String typeOf(float value) {
        return "float";
    }

    public static String typeOf(Float value) {
        return "Float";
    }

    public static String typeOf(long value) {
        return "long";
    }

    public static String typeOf(Long value) {
        return "Long";
    }

    public static String typeOf(short value) {
        return "short";
    }

    public static String typeOf(Short value) {
        return "Short";
    }

    public static String typeOf(byte value) {
        return "byte";
    }

    public static String typeOf(Byte value) {
        return "Byte";
    }

    public static String typeOf(char value) {
        return "char";
    }

    public static String typeOf(Character value) {
        return "Character";
    }

    public static String typeOf(boolean value) {
        return "boolean";
    }

    public static String typeOf(Boolean value) {
        return "Boolean";
    }

    // ... add more overloaded methods for other types if needed
}
```