Wrapper Classes in Java

Definition & Purpose:

- **Wrapper Classes** are predefined classes in the `java.lang` package that allow conversion between primitive data types (e.g., `int`, `float`) and their corresponding Object types (e.g., `Integer`, `Float`).
- They are **immutable**, meaning that once an object is created, its value cannot be changed. Any modification produces a new object.
- **Collections** in Java cannot store primitive types directly; they store object references. To work with primitives in Collections, you must convert them into their respective wrapper objects.

Why Use Wrapper Classes?

When working with Collections, follow these steps:

1. **Storing Primitive Data in a Collection:**
   - **Declare** the primitive.
   - **Convert** the primitive to its wrapper (Object) form.
   - **Store** the wrapper object in the Collection.
2. **Retrieving Primitive Data:**
   - **Retrieve** the Object from the Collection.
   - **Convert** the Object back to the primitive type.

Types of Conversions

Conversion Type	Method/Mechanism	Example Code
Primitive to Object	a. Parameterized constructor b. `valueOf()` method c. Auto-Boxing	See examples below
Object to Primitive	a. `xxxValue()` method b. Auto-Unboxing	See examples below
String to Object	a. String parameterized constructor b. `valueOf()` method	See examples below
Object to String	a. `toString()` method b. Using `+` operator	See examples below
Primitive to String	a. Static `toString()` method b. Using `+` operator	See examples below
String to Primitive	a. Static `parseXxx()` method	See example below

---

1. Conversions from Primitive Types to Object Types

- **a. Using Parameterized Constructor:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        int i = 10;
        Integer in = new Integer(i);
        System.out.println(i + "    " + in);
    }
}
// Output: 10    10
```

- **b. Using `valueOf()` Method:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        int i = 10;
        Integer in = Integer.valueOf(i);
        System.out.println(i + "    " + in);
    }
}
// Output: 10    10
```

- **c. Using Auto-Boxing (Java 1.5+):**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        int i = 10;
        Integer in = i; // Auto-boxing
        System.out.println(i + "    " + in);
    }
}
// Output: 10    10
```

---

2. Conversions from Object Types to Primitive Types

- **a. Using `xxxValue()` Method:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        Integer in = new Integer(10);
        int i = in.intValue();
        System.out.println(in + "    " + i);
    }
}
// Output: 10    10
```

- **b. Using Auto-Unboxing (Java 1.5+):**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        Integer in = new Integer(10);
        int i = in; // Auto-unboxing
        System.out.println(in + "    " + i);
    }
}
// Output: 10    10
```

---

3. Conversions from String Type to Object Types

- **a. Using String Parameterized Constructor:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        String str = new String("10");
        Integer in = new Integer(str);
        System.out.println(str + "    " + in);
    }
}
// Output: 10    10
```

- **b. Using `valueOf()` Method:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        String str = new String("10");
        Integer in = Integer.valueOf(str);
        System.out.println(str + "    " + in);
    }
}
// Output: 10    10
```

---

4. Conversions from Object Type to String Type

- **a. Using `toString()` Method:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        Integer in = new Integer(10);
        String str = in.toString();
        System.out.println(in + "    " + str);
    }
}
// Output: 10   10
```

- **b. Using the + Concatenation Operator:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        Integer in = new Integer(10);
        String str = "" + in; // Implicitly calls toString()
        System.out.println(in + "    " + str);
    }
}
// Output: 10   10
```

---

5. Conversions from Primitive Type to String Type

- **a. Using Static `toString()` Method:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        int i = 10;
        String str = Integer.toString(i);
        System.out.println(i + "    " + str);
    }
}
// Output: 10    10
```

- **b. Using the + Concatenation Operator:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        int i = 10;
        String str = "" + i;
        System.out.println(i + "    " + str);
    }
}
// Output: 10    10
```

---

6. Conversions from String Type to Primitive Type

- **a. Using Static `parseXxx()` Method:**

```
java
CopyEdit
public class Main {
    public static void main(String[] args) {
        String data = "10";
        int i = Integer.parseInt(data);
        System.out.println(data + "    " + i);
    }
}
// Output: 10    10
```

Additional Notes

- **Immutability:**
  Wrapper class objects are immutable. Any change creates a new object rather than modifying the existing one.
- **Usage in Collections:**
  Since Java Collections cannot store primitives directly, these conversions enable the storage and manipulation of numerical and other primitive values in collections like `ArrayList`, `HashSet`, etc.
- **Auto-Boxing & Auto-Unboxing:**
  Introduced in Java 1.5, these mechanisms allow automatic conversion between primitive types and their corresponding wrapper objects, making code more concise and readable.