Northeastern University

Lecture 9: Object Oriented Programming - 4

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Liang, Y. Daniel. Introduction to Java Programming, Comprehensive Version, 12th edition, Pearson, 2019.

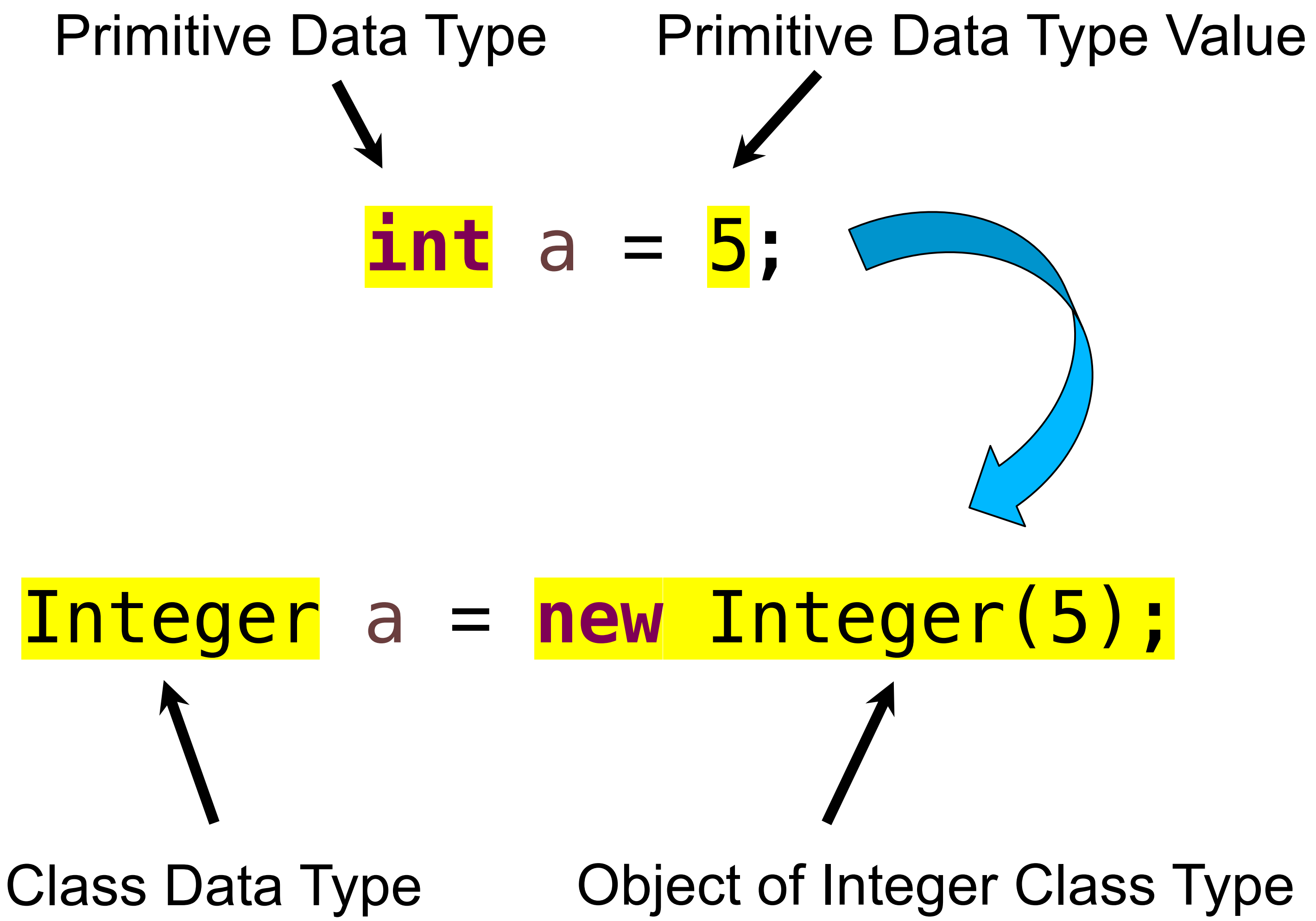
Outline

- Objects and Classes
- Thinking in Objects

Outline

- Objects and Classes
- Thinking in Objects

Processing Primitive Data Type Values as Objects



```
int a = 5;
```

```
Integer a = new Integer(5);
```

Wrapper Classes

- Byte
- Short
- Integer
- Long
- Float
- Double
- Boolean
- Character

NOTE:

(1) The wrapper classes do not have no-arg constructors.

(2) The instances of all wrapper classes are immutable, i.e., their internal values cannot be changed once the objects are created.

The `Integer` and `Double` Class

java.lang.Integer

```
-value: int
+MAX_VALUE: int
+MIN_VALUE: int

+Integer(value: int)
+Integer(s: String)
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longVlaue(): long
+floatValue(): float
+doubleValue():double
+compareTo(o: Integer): int
+toString(): String
+valueOf(s: String): Integer
+valueOf(s: String, radix: int): Integer
+parseInt(s: String): int
+parseInt(s: String, radix: int): int
```

java.lang.Double

```
-value: double
+MAX_VALUE: double
+MIN_VALUE: double

+Double(value: double)
+Double(s: String)
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longVlaue(): long
+floatValue(): float
+doubleValue():double
+compareTo(o: Double): int
+toString(): String
+valueOf(s: String): Double
+valueOf(s: String, radix: int): Double
+parseDouble(s: String): double
+parseDouble(s: String, radix: int): double
```

The `Integer` and `Double` Class

- Constructors
- Class Constants `MAX_VALUE`, `MIN_VALUE`
- Conversion Methods

Numeric Wrapper Class - Constructors

- You can construct a wrapper object either from a primitive data type value or from a string representing the numeric value.
- The constructors for `Integer` and `Double` are:

```
public Integer(int value)
public Integer(String s)
public Double(double value)
public Double(String s)
```

Numeric Wrapper Class - Constants

- Each numerical wrapper class has the constants `MAX_VALUE` and `MIN_VALUE`.
- `MAX_VALUE`:
  - maximum value of the corresponding primitive data type.
- `MIN_VALUE`:
  - For `Byte`, `Short`, `Integer`, and `Long`, it represents the minimum `byte`, `short`, `integer`, and `long` values.
  - For `Float` and `Double`, it represents the minimum *positive* `float` and `double` values.

Conversion Methods

- Each numeric wrapper class implements the abstract methods doubleValue, floatValue, intValue, longValue, and shortValue, which are defined in the `Number` class.
- These methods “convert” objects into primitive type values.

Exercise

- Please ask the user to enter an integer and use this input value to create an `Integer` object.
- Print out this `Integer` object’s value.

Answer

```
import java.util.Scanner;

public class ClassExercise {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        System.out.print("Please enter an integer: ");

        int i = input.nextInt();

        Integer value = new Integer(i);

        System.out.print("The input value is " + value.intValue());

        input.close();

    }

}
```

The Static `valueOf` Methods

- The numeric wrapper classes have a useful class (static) method, `valueOf(String s)`.
- This method creates a new object initialized to the value represented by the specified string. For example:

```
Double doubleObject = Double.valueOf("12.4");

Integer integerObject = Integer.valueOf("12");
```

The Methods for Parsing Strings into Numbers

- We can use the static `parseInt` method in the `Integer` class to parse a numeric string into an int value and the static `parseDouble` method in the `Double` class to parse a numeric string into a double value.

- Each numeric wrapper class has two overloaded parsing methods to parse a numeric string into an appropriate numeric value.

```
// These two methods are in the Integer class
public static int parseInt(String s)
public static int parseInt(String s, int radix)
```

Exercise

- Type in following code and see what is the output of the following code?

```
public class ClassExercise {

    public static void main(String[] args){

        System.out.println(Integer.parseInt("14"));
        System.out.println(Integer.parseInt("14",10));
        System.out.println(Integer.parseInt("14",16));

        System.out.println(Integer.parseInt("13"));
        System.out.println(Integer.parseInt("13",10));
        System.out.println(Integer.parseInt("13",16));

    }

}
```

Answer

```
14
14
20
13
13
19
```

Exercise

- Assume you have a `Double` class type variable x. The x value is 3.3.

```
Double x = 3.3;
```

- Could you print out the real number part by using a `Double` class method?

- Could you compare x value with 5.5 by using a `Double` class method? If they are the same, output 0. Otherwise, output -1.

Answer

```
public class ClassExercise {
    public static void main(String[] args){
        Double x = 3.3;
        System.out.println(x.intValue());
        System.out.println(x.compareTo(5.5));
    }
}
```

output:

```
3
-1
```

The `String` Class

The `String` Class

- Constructing a String:

```
String message = "Welcome to Java";
String message = new String("Welcome to Java");
String s = new String();
```

- Can obtain String length (`.length()`) and retrieve individual characters (`.charAt(index)`)
- String Concatenation: `concat`
- Substrings: `substring(index)`, `substring(start, end)`
- Comparisons: `equals`, `compareTo`

The `String` Class (cont.)

- String Conversions (between Strings and Arrays):

  `.toCharArray()`

```
        char[] charArray = s.toCharArray();
```

- Converting characters and numeric values to Strings:

  `.valueOf(char)`, `.valueOf(int)`

Constructing Strings

```
String newString = new String(stringLiteral);

String message = new String("Welcome to Java”);
```

- Since strings are used frequently, Java provides a shorthand initializer for creating a string:

```
String message = "Welcome to Java";
```

Strings Are Immutable

- A String object is immutable; its contents cannot be changed. Does the following code change the contents of the string?

```
String s = "Java";

s = "HTML";
```

- The answer is no. Why?