Wrapper Classes for the Primitives Types

Prepared by

Dr. Rajesh Kumar Ojha
Asst. Prof., CSE, Silicon University

Primitives & Wrappers

- Java has a *wrapper* class for each of the eight primitive data types:

Primitive Type	Wrapper Class	Primitive Type	Wrapper Class
boolean	Boolean	float	Float
byte	Byte	int	Integer
char	Character	long	Long
double	Double	short	Short

Use of the Wrapper Classes

- Java's *primitive* data types (boolean, int, etc.) are not classes.
- Wrapper classes are used in situations where objects are required, such as for elements of a Collection:

```
List<Integer> a = new ArrayList<Integer>();
methodRequiringListOfIntegers(a);
```

Value => Object: Wrapper Object Creation

- ***Wrapper*.valueOf()** **takes a value (or string) and returns an object of that class:**

```
Integer i1 = Integer.valueOf(42);
Integer i2 = Integer.valueOf("42");

Boolean b1 = Boolean .valueOf(true);
Boolean b2 = Boolean .valueOf("true");

Long n1 = Long.valueOf(42000000L);
Long n1 = Long.valueOf("42000000L");
```

Object => Value

- Each wrapper class Type has a method typeValue to obtain the object’s value:

```
Integer i1 = Integer.valueOf(42);
Boolean b1 = Boolean.valueOf(“false”);
System.out.println(i1.intValue());
System.out.println(b1.intValue());

=>
42
false
```

String => value

- The Wrapper class for each primitive *type* has a method parse*Type*() to parse a string representation & return the literal value.

```
Integer.parseInt("42")          => 42
Boolean.parseBoolean("true")    => true
Double.parseDouble("2.71")      => 2.71
//…
```

- Common use: Parsing the arguments to a program:

Parsing argument lists

```
// Parse int and float program args.
public parseArgs(String[] args) {
  for (int i = 0; i < args.length; i++) {
   try {
     …println(Integer.parseInt(args[i]));
   } catch (Exception e) {
   try {
     …println(Float.parseFloat(args[i]));
   } finally { }
}}}
```

Parsing argument lists

```
=>
arg # 0 = 0

arg # 1 = 42

arg # 2 = 999

arg # 3 = 0.0

arg # 4 = 1.42

arg # 5 = 9.0008
```

Sample values:

```
boolObj new Boolean(Boolean.TRUE);
charObj = new Character('a');
byteObj = new Byte("100");
shortObj = new Short("32000");
intObj = new Integer(2000000);
longObj = new Long(500000000000000000L);
floatObj = new Float(1.42);
doubleObj = new Double(1.42);

printWrapperInfo(); //method to print objects above
```

Sample values (output from previous slide):

```
=>
For Boolean & Character Wrappers:
    Boolean:true
    Character:a

For Number wrappers:
    Byte:100
    Short:32000
    Integer:2000000
    Long:5000000000000000000
    Float:1.42
    Double:1.42
```

Each Number Wrapper has a MAX_VALUE constant:

```
byteObj = new Byte(Byte.MAX_VALUE);
shortObj = new Short(Short.MAX_VALUE);
intObj = new Integer(Integer.MAX_VALUE);
longObj = new Long(Long.MAX_VALUE);
floatObj = new Float(Float.MAX_VALUE);
doubleObj = new Double(Double.MAX_VALUE);

printNumValues("MAXIMUM NUMBER VALUES:");
```

MAX values (output from previous slide):

```
=>
   Byte:127
   Short:32767
   Integer:2147483647
   Long:9223372036854775807
   Float:3.4028235E38
   Double:1.7976931348623157E308
```

Many useful utility methods, e.g., for Integer:

```
       int    hashCode()
static int    numberOfLeadingZeros(int i)
static int    numberOfTrailingZeros(int i)
static int    reverse(int i)
static int    reverseBytes(int i)
static int    rotateLeft(int i, int distance)
static int    rotateRight(int i, int distance)
static String toBinaryString(int i)
static String toHexString(int i)
static String toOctalString(int i)
static String toString(int i, int radix)
```

Double & Float: Utilities for Arithmetic Operations:

- Constants POSITIVE_INFINITY & NEGATIVE_INFINITY
- Constant NaN = Not-a-Number (NaN) value.
- Methods isNaN(), isInfinite()

Thank you