Java Lec 4 Wrapper Object

Monday, August 23, 2021 8:57 PM

Java_Wra...

Object Oriented Programming

Wrapper classes, Java Type System, The Object class

Library

int data

Primitive Types

for(int i=1; i<5; i++)

Type	Description	Size
`int`	The integer type, with range –2,147,483,648 . . . 2,147,483,647	4 bytes
`byte`	The type describing a single byte, with range –128 . . . 127	1 byte
`short`	The short integer type, with range –32768 . . . 32767	2 bytes

`long`	The long integer type, with range –9,223,372,036,854,775,808 . . . –9,223,372,036,854,775,807	8 bytes

Continued…

Primitive Types

Type	Description	Size
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme	2 bytes
`boolean`	The type with the two truth values `false` and `true`	1 byte

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

int arrays [int, int, float, char, boolean] Object array [ ] [ ] [ ] [ ] Student Name, DOB, CGPA

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);
```

5

Use of the Wrapper Classes: Methods for conversion

- Value => Object valueOf

- Value => Object valueOf

- Object => Value

  intValue, booleanValue

- String => value

  parseInt, parseBoolean, …

toString()

6

Value => Object: Wrapper Object Creation

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

int

```
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");
```

objects.

main String[] args

7

behaviour → System.out

Object => Value

Object => value

Object Array

→ Rectangle
x y w h
int int int int

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

object

i=42;

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

true false

1

```
=>
42
false
```

G1

Integer Object

(i1);

O/P mean

String => value

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

class method

String → int → Integer Object

```
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++) {

    …println(Integer.parseInt(args[i]));
}
}
```

int

"String"

parseX

10

Many useful utility methods: Integer

G3

int

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)
```

G2 examples

11

Number

Double & Float: Utilities for Arithmetic Operations:

portable

→ Math.PI

pi

- Constants

→ POSITIVE_INFINITY, NEGATIVE_INFINITY

- Constant NaN = Not-a-Number (NaN) value.

- Methods isNaN(), isInfinite()

G4

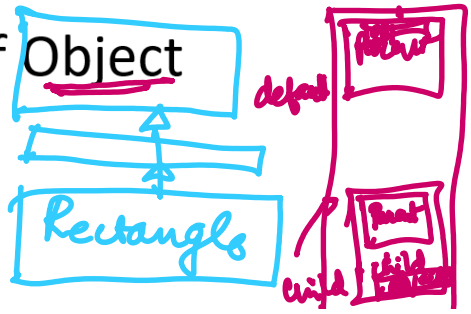
12

Class Object

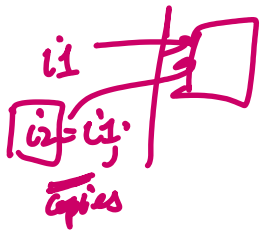
Name of a class

object of a class

- **`Object`** is the root of the class hierarchy
  - Every *class* has **`Object`** as a superclass
- All classes inherit the methods of Object
  - But may override them



Class `Object`



equals. shallow
deeper comparison

TABLE 3.2
Methods of Class `java.lang.Object`

Method	Behavior
`Object clone()`	Makes a copy of an object.
`boolean equals(Object obj)`	Compares this object to its argument.
`int hashCode()`	Returns an integer hash code value for this object.
`String toString()`	Returns a string that textually represents the object.

G5 →
Rectangle
Student

how does it actually compare
better X,Y OR W H OR 1 more way
→ customized

The Method `toString`

- You should always override `toString` method if you want to print object state

- If you do *not* override it:

If you do not override it:

- **`Object.toString`** will return a **`String`**
- Just not the **`String`** you want!

  Example: **`ArrayBasedPD@ef08879`**

  ... The name of the class, `@`, instance's hash code

15

Always override toString()

"When practical, the toString method should return all of the interesting info contained in the object."

Note that toString should never print anything

toString() called automatically

System.out.println("Answer = " + 42);

toString() method is

called automatically

```
System.out.println( d1 );

System.out.println( d1.topFace() );

System.out.println( d1.toString() );
```

unnecessary, adds clutter