OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

08 – Java Data Types (Часть 4)

Дмитрий Коган

Как меня слышно и видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели :

- Займёмся чтением и записью полей объекта
- Изучим классы-оболочки
- Составим «волшебный треугольник»

Начинаем?

Темы экзамена

- ❑ Java Basics
- ❑ Working with Java Data Types
- ❑ Using Operators and Decision Constructs
- ❑ Creating and Using Arrays
- ❑ Using Loop Constructs
- ❑ Working with Methods and Encapsulation
- ❑ Working with Inheritance
- ❑ Handling Exceptions
- ❑ Working with Selected classes from the Java API

Подтемы экзамена

Working with Java Data Types

- Declare and initialize variables (including casting of primitive data types)
- Differentiate between object reference variables and primitive variables
- Know how to read or write to object fields
- Explain an object's lifecycle (creation, "dereference by reassignment" and garbage collection)
- Develop code that uses wrapper classes such as Boolean, Double and Integer

Чтение и запись полей объекта

Объектное поле

- «Объектное поле» (object field) – это просто другое название для переменной экземпляра (instance variable), определенной в том или ином классе.

- Доступ к объектным полям:
  - Чтение:

    – напрямую по имени данного поля (если позволяет уровень доступа), или
    – посредством метода, который возвращает значение этого поля.

  - Запись:

    – напрямую по имени данного поля (если позволяет уровень доступа), или
    – через конструктор или метод, который присваивает полю некое значение.

Упражнение

```
class ObjectFields {
    int iVar;
    static int cVar;
    void setFields(){
        this.iVar = 22;
        this.cVar = 22;
    }
    public static void main(String[] args) {
        ObjectFields of = new ObjectFields();
        of.iVar = 100;
        ObjectFields.cVar = 200;
        iVar = 200;
        cVar = 300;
        this.iVar = 200;
        this.cVar = 400;
        of.iVar = 100;
        of.cVar += 200;
        ObjectFields.cVar += 300;
        of.setFields();
        System.out.println("iVar=" + of.iVar + ", cVar=" + of.cVar);
    }
}
```

What is the result if we comment out all those LOCs that do not compile?

A. iVar=22, cVar=22
B. iVar=200, cVar=300
C. iVar=400, cVar=700
D. iVar=600, cVar=1100

Ответ: А

Упражнение

```
class Q {
    String str = "null";
    int a = 12;
    Q(String String) {
        str = String;
    }
    Q(int a) {
        this.a = a;
    }
    void println() {
        System.out.print(str + " " + a + " ");
    }
    public static void main(String[] args) {
        new Q("Hi!").println();
        new Q(123).println();
    }
}
```

What is the result?

A. null 12 null 123
B. Hi! 12 null 12
C. Hi! 12 null 123
D. Compilation fails

Ответ: C

Упражнение

```
class Engine {
    int hp;
    String model;
    Engine() {
        hp = 80;
        model = "Junior";
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Engine e1 = new Engine();
        Engine e2 = test1(e1);
        test2(e2);
        System.out.println(e1.hp + " " + e2.model);
    }
    public static Engine test1(Engine e) {
        e.model = "Senior";
        e.hp = 120;
        return e;
    }
    public static void test2(Engine e) {
        e.model = "Grandpa";
        e.hp = 170;
        return;
    }
}
```

What is the result?

A. 80 Junior
B. 120 Senior
C. 170 Grandpa
D. Compilation fails

Ответ: C

Вопросы?

Классы-оболочки

Определение

Для каждого примитивного типа Джава предоставляет по специализированному классу:

`byte` ↔ **Byte**, `short` ↔ **Short**, `int` ↔ **Integer**, `long` ↔ **Long**,

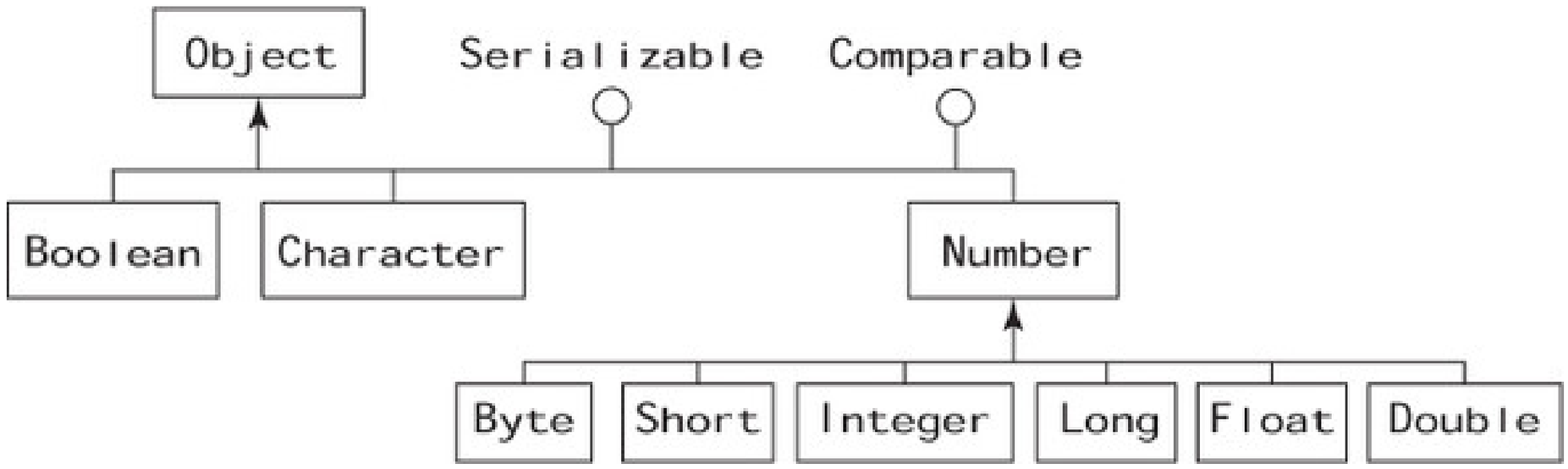
`float` ↔ **Float**, `double` ↔ **Double**,

`char` ↔ **Character**,

`boolean` ↔ **Boolean**

Это и есть классы-оболочки.

Иерархия классов



- Все оболочечные классы являются немутирующими.
- Хотя все «рапперы» расширяют **Number**, между самими оболочками нет отношений IS-A. То есть, попытка скастировать **Integer** к **Long** не сработает. **Byte** == **Short** и т.п. тоже не проходят компиляцию.

Зачем нужны?

Порой с примитивом куда проще справиться, когда он ведет себя подобно объекту. Например, классы-оболочки предоставляют множество полезных методов, но все они хотят, чтобы их аргументами был какой-то ссылочный тип. А иногда без этого просто не обойтись: скажем, будучи структурой данных типа **Collection**, класс **ArrayList** вообще не может содержать примитивы. В этом случае Джава способна автоупаковать переданные примитивные аргументы в соответствующие «оболочечные» типы – и уже после этого мы сможем воспользоваться удобными методами из пакета **java.util**, например, **Collections.sort()** и т.п.

Автоупаковка

Автоупаковкой называется процесс преобразования примитивных дата-типов в объекты, другими словами, в экземпляры (instances) соответствующих классов-оболочек. Такая «упаковка» автоматически выполняется самим компилятором.

Распаковка

Автоматическое преобразование объекта-оболочки в соответствующий примитивный тип называется распаковкой.

Конструкторы

Primitive type	Wrapper class	Example of constructing
boolean	Boolean	`new Boolean(true)`
byte	Byte	`new Byte((byte) 1)`
short	Short	`new Short((short) 1)`
int	Integer	`new Integer(1)`
long	Long	`new Long(1)`
float	Float	`new Float(1.0)`
double	Double	`new Double(1.0)`
char	Character	`new Character('c')`

Конструкторы

- Никто из оболочек не имеет безаргументного (no-arg, zero-arg) конструктора.
- ВСЕ оболочки (кроме **Character**) принимают **String** и ВСЕ (без исключений) принимают свои «родные» примитивы.

```
Boolean bool2 = new Boolean(true);
Character char2 = new Character('a');
Byte byte2 = new Byte((byte)10);
Double double2 = new Double(10.98);

//Character char3 = new Character("a");
Boolean bool3 = new Boolean("true");
Byte byte3 = new Byte("10");
Double double3 = new Double("10.98");
```

Constructors that accept primitive value

Won't compile (if uncommented)

Constructor that accepts String

Конструкторы

- Конструкторы всех оболочек КРОМЕ **Character** (где есть лишь один-единственный конструктор: **Character**(char value)) принимают `null`, считая его просто null-литералом типа **String**, хотя затем бросают NumberFormatException (или NPE в случае **Float** и **Double**), ЗА ИСКЛЮЧЕНИЕМ конструктора **Boolean**(null), который возвращает объектную константу `FALSE` (не путать с примитивным `false`).

Конструкторы

Конструкторы: обычно по 2 конструктора (кроме **Character** и **Float**)

Boolean(boolean value)
Boolean(String s) ← безразличен к регистру букв (case-insensitive)!
важнее всего: если String == **null** ИЛИ **не отражает false/true:**
→ Boolean.FALSE (т.е. объект, соответствующий `false`-примитиву)

Integer(int value) - - - - - - } здесь ловушек нет
Integer(String s) - - - - - - }

Byte(byte value) ← КАПКАН: берёт только byte! new Byte(1) не скомп.; нужен каст
Byte(String s) ← КАПКАН: new Byte("128") бросит NFE

Short(short value) - - - - - - } берёт и `short` и `byte` (будет расширен)
Short(String s) - - - - - - }

Long(long value) - - - - - - } берёт любой целочисленный тип, кроме `char`
Long(String s) - - - - - - }

Character(char value) ← берёт лишь ‘апострофные’ аргументы, напр., 'a' или '\u0123'
или делаем каст, напр., Character c = new Character((char)1);

Float(double value) ← удивительно, но факт… syntax sugar?
Float(float value)
Float(String s) ← Float("1"), Float("1f") и Float("1d") годятся
Float("1L") бросает NFE
Float("01"), Float("01F") и Float("01D") годятся
Float("0x…[F/D]") и Float("0b…") бросают NFE

Double(double value)
Double(String s) ← аналогично случаю Float

Популярные методы

valueOf(): STATIC! вернет ***Wrapper***-объект; принимает и **String** и «родной» примитив

static Integer valueOf(int i)	Вернет **Integer**-объект, соответствующий указанному `int`-аргументу.
static Integer valueOf(String s)	Вернет **Integer**-объект, соответствующий указанному **String**-аргументу.

***prim*Value():** НЕ STATIC! вернет «родной» примитив; это безаргументный метод;

int intValue()	Вернет значение этого **Integer**-объекта в виде `int`.

parse*Prim*(): STATIC! вернет примитив, соотв. **String**-аргументу; парсеры берут лишь стринги

static int parseInt(String s)	Парсирует стринг, считая его десятичным целым со знаком.

Популярные методы

```
int primitive = Integer.parseInt("123");
Integer wrapper = Integer.valueOf("123");
```

```
int bad1 = Integer.parseInt("a");              // throws NumberFormatException
Integer bad2 = Integer.valueOf("123.45");      // throws NumberFormatException
```

- Парсеры классов-оболочек принимают лишь **String** (или **String**, radix).

Популярные методы

```
Boolean bool4 = Boolean.valueOf(true);
Boolean bool5 = Boolean.valueOf(true);
Boolean bool6 = Boolean.valueOf("TrUE");
Double double4 = Double.valueOf(10);
```

Using static method valueOf()

```
Long.parseLong("12.34");
```

Throws NumberFormatException: 12.34 isn't a valid long

```
Byte.parseByte("1234");
```

Throws NumberFormatException: 1234 is out of range for byte

```
Boolean.parseBoolean("true");
```

Returns boolean true

```
Boolean.parseBoolean("TrUe");
```

No exceptions; the String argument isn't case-sensitive

Конвертация строк

Wrapper class	Converting `String` to primitive	Converting `String` to wrapper class
Boolean	`Boolean.parseBoolean("true");`	`Boolean.valueOf("TRUE");`
Byte	`Byte.parseByte("1");`	`Byte.valueOf("2");`
Short	`Short.parseShort("1");`	`Short.valueOf("2");`
Integer	`Integer.parseInt("1");`	`Integer.valueOf("2");`
Long	`Long.parseLong("1");`	`Long.valueOf("2");`
Float	`Float.parseFloat("1");`	`Float.valueOf("2.2");`
Double	`Double.parseDouble("1");`	`Double.valueOf("2.2");`
Character	None	None

Упражнение

```
public class BoolArray {
    public static void main(String[] args) {
        Boolean[] barr = new Boolean[3];
        barr[0] = new Boolean(Boolean.parseBoolean("tRUE"));
        barr[1] = new Boolean("True");
        barr[2] = new Boolean(false);
        System.out.println(barr[0] + " " + barr[1] + " " + barr[2]);
    }
}
```

What is the result?

A. true false false
B. true true false
C. false false false
D. An IllegalArgumentException is thrown at run time
E. A NullPointerException is thrown at run time

Ответ: В

Boolean

- **Boolean.valueOf(**String**)** и его перегруженный вариант **Boolean.valueOf(**boolean**)** возвращает ссылку на **Boolean.**TRUE или **Boolean.**FALSE, вовсе не создавая новый оболочечный объект; другими словами, эти методы всего лишь возвращают `static`-константы **TRUE** или **FALSE**, определенные в классе **Boolean**.

- **Boolean**.FALSE – это **Object** (то же самое и в случае **Boolean**.TRUE).

Численный пул

- **Byte**, **Short**, **Integer** и **Long** пользуются одинаковым численным пулом (аналогично стрингам) в пределах байта (-128 ... 127), вот почему оператор == даёт `true`, когда у разных объектов одинаковые численные значения в указанных границах. Ну а вне этих границ оператор ‘double equals’ сравнивает уже ссылки на объекты.

  - `Character` from `\u0000` to `\u007f` (`7f` is `127` in decimal)

Численный пул

```
Long var1 = Long.valueOf(123);
Long var2 = Long.valueOf("123");
System.out.println(var1 == var2);
```

Prints true; var1 and var2 refer to the same cached object.

```
Long var3 = Long.valueOf(223);
Long var4 = Long.valueOf(223);
System.out.println(var3 == var4);
```

Prints false; var3 and var4 refer to different objects.

Численный пул

```
Integer i1 = new Integer(10);
Integer i2 = new Integer(10);
```

Constructors always create new instances.

```
Integer i3 = Integer.valueOf(10);
Integer i4 = Integer.valueOf(10);
```

valueOf returns a cached copy for int value 10.

```
Integer i5 = 10;
Integer i6 = 10;
```

Autoboxing returns a cached copy for applicable values.

```
System.out.println(i1 == i2);
System.out.println(i3 == i4);
System.out.println(i4 == i5);
System.out.println(i5 == i6);
```

Численный пул

```
Integer i1 = new Integer(10);
Integer i2 = new Integer(10);
```

Constructors always create new instances.

```
Integer i3 = Integer.valueOf(10);
Integer i4 = Integer.valueOf(10);
```

valueOf returns a cached copy for int value 10.

```
Integer i5 = 10;
Integer i6 = 10;
```

Autoboxing returns a cached copy for applicable values.

```
System.out.println(i1 == i2);
System.out.println(i3 == i4);
System.out.println(i4 == i5);
System.out.println(i5 == i6);
```

```
false
true
true
true
```

Численный пул

```
System.out.println(i1.equals(i2));
System.out.println(i3.equals(i4));
System.out.println(i4.equals(i5));
System.out.println(i5.equals(i6));
```

Output
true

Численный пул

```
Integer i1 = new Integer(200);
Integer i2 = new Integer(200);

Integer i3 = Integer.valueOf(200);
Integer i4 = Integer.valueOf(200);

Integer i5 = 200;
Integer i6 = 200;

System.out.println(i1 == i2);
System.out.println(i3 == i4);
System.out.println(i4 == i5);
System.out.println(i5 == i6);
```

Return false—no cached copies for int value 200

Разные классы

```
Integer obj1 = 100;
Short obj2 = 100;

System.out.println(obj1.equals(obj2));
System.out.println(obj1 == obj2);
```

Outputs false

Doesn't compile

Упражнение

```
public class RQ200A70 {
  public static void main(String[] args) {
    Integer i = new Integer(-10);
    Integer j = new Integer(-10);
    Integer k = -10;
    System.out.print((i==j) + "|");
    System.out.print(i.equals(j) + "|");
    System.out.print((i==k) + "|");
    System.out.print(i.equals(k));
  }
}
```

Select the one correct answer.

(a) `false|true|false|true`

(b) `true|true|true|true`

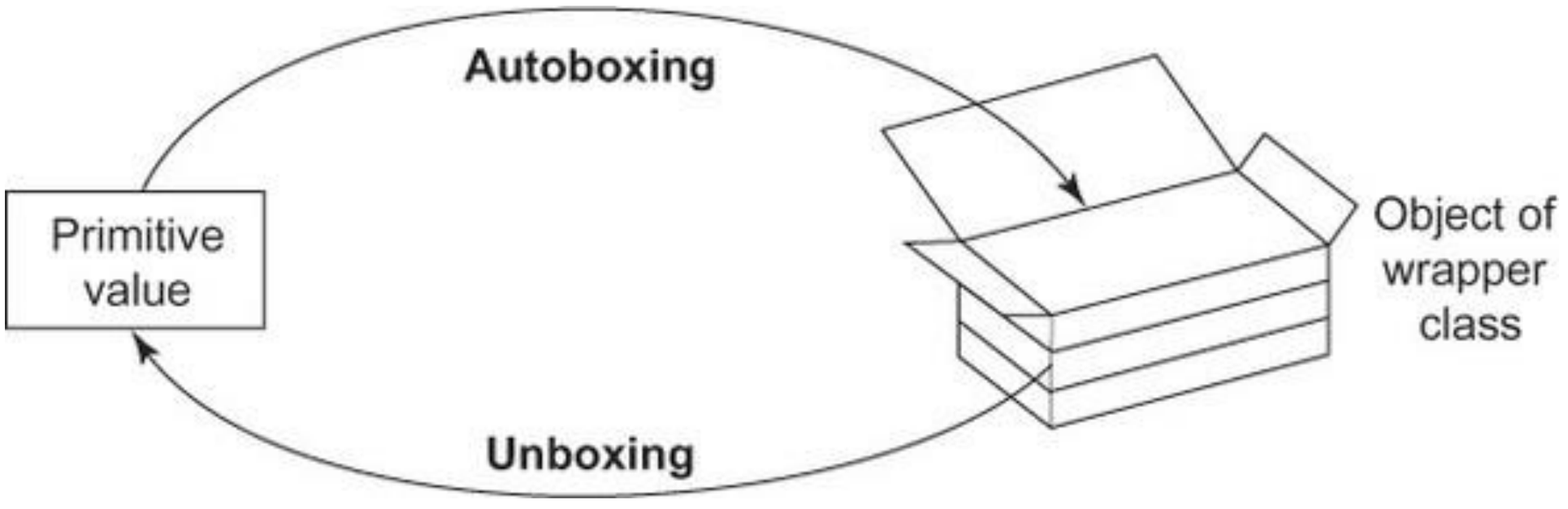
(c) `false|true|true|true`

(d) `true|true|false|true`

(e) None of the above.

Ответ: А

Упаковка


Autoboxing
Primitive value
Object of wrapper class
Unboxing

Пакуем налету

```
Double d1 = new Double(12.67);
System.out.println(d1.compareTo(21.68));
```

Prints -1, since 12.67 < 21.68

```
public int compareTo(Double anotherDouble)
```

```
Double d1 = new Double(12.67D);
System.out.println(d1.compareTo(Double.valueOf(21.68D)));
```

Пакуем налету

```
public class Unboxing {
    public static void main (String args[]) {
        ArrayList<Double> list = new ArrayList<Double>();
        list.add(12.12);
        list.add(11.24);
        Double total = 0.0;
        for (Double d : list)
            total += d;
    }
}
```

List of Double

Autoboxing—add double

Unbox to use operator += with total

```
public class Unbox {
    public static void main(String args[]) {
        ArrayList list = new ArrayList();
        list.add(new Double(12.12D));
        list.add(new Double(11.24D));
        Double total = Double.valueOf(0.0D);
        for(Iterator iterator = list.iterator(); iterator.hasNext();) {
            Double d = (Double)iterator.next();
            total = total.doubleValue() + d.doubleValue();
        }
    }
```

Немутирующие классы

Все оболочечные классы являются немутирующими.

```
Double total = Double.valueOf(0.0D);
total = total.doubleValue() + d.doubleValue();
```

До Java 5 и после

```
Integer y = new Integer(567);     // make it
int x = y.intValue();             // unwrap it
x++;                              // use it
y = new Integer(x);               // rewrap it
System.out.println("y = " + y);   // print it
```

```
Integer y = new Integer(567);     // make it
y++;                              // unwrap it, increment it,
                                  // rewrap it
System.out.println("y = " + y);   // print it
```

Ссылки и объекты

```
Integer y = 567;                    // make a wrapper
Integer x = y;                      // assign a second ref
                                    // var to THE wrapper

System.out.println(y==x);           // verify that they refer
                                    // to the same object
y++;                                // unwrap, use, "rewrap"
System.out.println(x + " " + y);    // print values

System.out.println(y==x);           // verify that they refer
                                    // to different objects

true
  567 568
  false

int x2 = y.intValue();              // unwrap it
x2++;                               // use it
y = new Integer(x2);                // rewrap it
```

Работаем с коллекцией

```
4: List<Double> weights = new ArrayList<>();
5: weights.add(50.5);                    // [50.5]
6: weights.add(new Double(60));          // [50.5, 60.0]
7: weights.remove(50.5);                  // [60.0]
8: double first = weights.get(0);        // 60.0
```

```
3: List<Integer> heights = new ArrayList<>();
4: heights.add(null);
5: int h = heights.get(0);              // NullPointerException
```

NullPointerException

```
Integer integer;
integer++;                  // NPE
```

```
Integer integer = 2;
integer++;                  // 3
```

Работаем с коллекцией

```
List<Integer> numbers = new ArrayList<>();
numbers.add(1);
numbers.add(2);
numbers.remove( index: 1);
System.out.println(numbers);
```

Работаем с коллекцией

```
List<Integer> numbers = new ArrayList<>();
numbers.add(1);
numbers.add(2);
numbers.remove( index: 1);
System.out.println(numbers);     // [1]
numbers.remove(new Integer( value: 1));
System.out.println(numbers);     // []
```

Области применения

```
class UseBoxing {
  public static void main(String [] args) {
    UseBoxing u = new UseBoxing();
    u.go(5);
  }
  boolean go(Integer i) {          // boxes the int it was passed
    Boolean ifSo = true;           // boxes the literal
    Short s = 300;                 // boxes the primitive
    if(ifSo) {                     // unboxing
      System.out.println(++s);     // unboxes, increments, reboxes
    }
    return !ifSo;                  // unboxes, returns the inverse
  }
}
```

NullPointerException

```
class Boxing2 {
  static Integer x;
  public static void main(String [] args) {
    doStuff(x);
  }
  static void doStuff(int z) {
    int z2 = 5;
    System.out.println(z2 + z);
} }
```

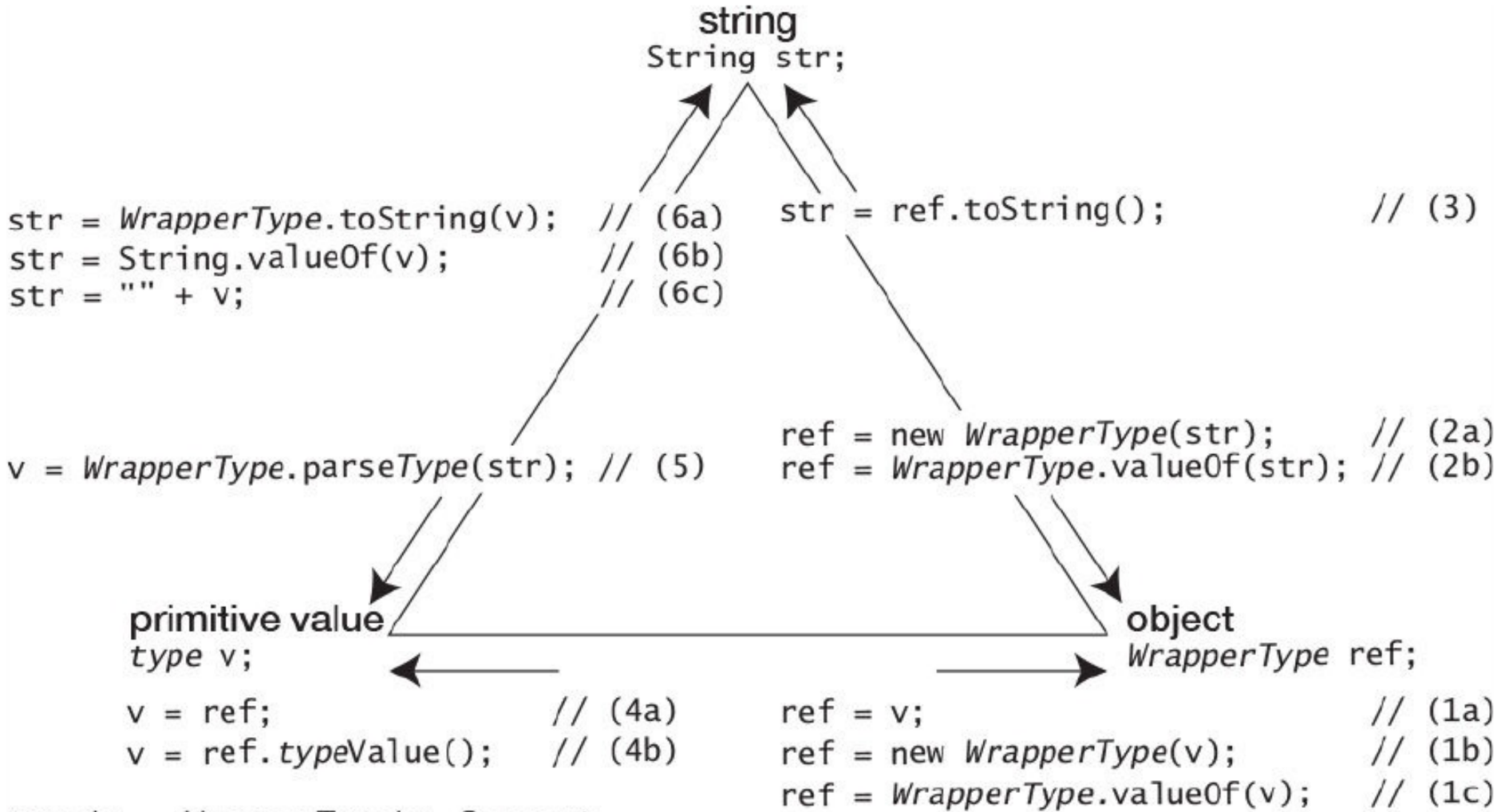
Ишь чего удумал!

Примитив, переданный в **List**<Wrapper> будет распакован на лету.

С другой стороны – ВНИМАНИЕ! ОЧЕНЬ ВАЖНОЕ ПРАВИЛО, КОТОРОЕ НАДО ПОМНИТЬ ВСЕГДА! – по собственному почину компилятор выполняет лишь одноэтапное (single-step) преобразование! К примеру, мы не можем добавить `byte` или `short` в **List**<Integer>:

```
byte b = 10;
short s = 20;
List<Integer> l = new ArrayList<>();
//    l.add(b);                          // не скомпилируется
//    l.add(s);                          // и этот LOC тоже не скомпилируется
l.add( (int)b );                         // VALID
```

Волшебный треугольник



type is:	*WrapperType is*:	*Comments:*
`boolean`	`Boolean`	(1a) Boxing.
`char`	`Character`	(2a) Not for `Character` type. Can throw `NumberFormatException`.
`byte`	`Byte`	(2b) Not for `Character` type. Can throw `NumberFormatException`.
`short`	`Short`	(4a) Unboxing.
`int`	`Integer`	(5) For numeric wrapper types and `Boolean` only.
`long`	`Long`	Can throw `NumberFormatException` for numeric wrapper types.
`float`	`Float`	(6b) Not for `byte` and `short` primitive types.
`double`	`Double`	

(1a), (1b), (1c)

```
Character charObj1    = '\n';
Boolean   boolObj1    = true;
Integer   intObj1     = 2014;
Double    doubleObj1  = 3.14;
```

```
Character charObj1    = new Character('\n');
Boolean   boolObj1    = new Boolean(true);
Integer   intObj1     = new Integer(2014);
Double    doubleObj1  = new Double(3.14);
```

```
Character charObj1    = Character.valueOf('\n');
Boolean   boolObj1    = Boolean.valueOf(true);
Integer   intObj1     = Integer.valueOf(2014);
Double    doubleObj1  = Double.valueOf(3.14);
```

(2a)

```
Boolean boolObj2   = new Boolean(“TrUe”);        // case ignored: true
Boolean boolObj3   = new Boolean(“XX”);          // false
Integer intObj2    = new Integer(“2014”);
Double  doubleObj2 = new Double(“3.14”);
Long    longObj1   = new Long(“3.14”);             // NumberFormatException
```

(6b)

```
Boolean boolObj4    = Boolean.valueOf(“false”);
Integer intObj3     = Integer.valueOf(“1949”);
Double  doubleObj3 = Double.valueOf(“-3.0”);
```

```
Byte    byteObj1  = Byte.valueOf(“1010”, 2);    // Decimal value 10
Short   shortObj2 = Short.valueOf(“12”, 8);     // Not “\012”. Decimal value
10.
Integer intObj4   = Integer.valueOf(“-a”, 16); // Not “-0xa”. Decimal value
-10.
Long    longObj2  = Long.valueOf(“-a”, 16);     // Not “-0xa”. Decimal value
-10L.
```

(3), (6a)

```
String charStr    = charObj1.toString();      // "\n"
String boolStr    = boolObj2.toString();      // "true"
String intStr     = intObj1.toString();       // "2014"
String doubleStr = doubleObj1.toString();    // "3.14"
```

```
String charStr2    = Character.toString('\n');  // "\n"
String boolStr2    = Boolean.toString(true);    // "true"
String intStr2     = Integer.toString(2014);    // Base 10. "2014"
String doubleStr2 = Double.toString(3.14);      // "3.14"
```

(4a), (4b)

```
char    c = charObj1;                  // '\n'
boolean b = boolObj2;                  // true
int     i = intObj1;                   // 2014
double  d = doubleObj1;                // 3.14

char    c = charObj1.charValue();      // '\n'
boolean b = boolObj2.booleanValue();   // true
int     i = intObj1.intValue();        // 2014
double  d = doubleObj1.doubleValue();  // 3.14
```

compareTo(), equals()

```
// Comparisons based on objects created earlier
Character charObj2   = 'a';
int result1 = charObj1.compareTo(charObj2);       //  result1 < 0
int result2 = intObj1.compareTo(intObj3);         //  result2 > 0
int result3 = doubleObj1.compareTo(doubleObj2);   // result == 0
int result4 = doubleObj1.compareTo(intObj1);      // Compile-time error!
```

```
// Comparisons based on objects created earlier
boolean charTest   = charObj1.equals(charObj2);       // false
boolean boolTest   = boolObj2.equals(Boolean.FALSE); // false
boolean intTest    = intObj1.equals(intObj2);         // true
boolean doubleTest = doubleObj1.equals(doubleObj2);  // true
boolean test       = intObj1.equals(new Long(2014)); // false. Not same type.
```

MIN_VALUE, MAX_VALUE

```
byte   minByte    = Byte.MIN_VALUE;      // -128
int    maxInt     = Integer.MAX_VALUE;   // 2147483647
double maxDouble  = Double.MAX_VALUE;    // 1.7976931348623157e+308
```

(4b)

```
Byte    byteObj2   = new Byte((byte) 16);        // Cast mandatory
Integer intObj5    = new Integer(42030);
Double  doubleObj4 = new Double(Math.PI);

short  shortVal  = intObj5.shortValue();         // (1)
long   longVal   = byteObj2.longValue();
int    intVal    = doubleObj4.intValue();        // (2) Truncation
double doubleVal = intObj5.doubleValue();
```

(5)

```
byte   value1 = Byte.parseByte(“16”);
int    value2 = Integer.parseInt(“2010”);        // parseInt, not parseInteger
int    value3 = Integer.parseInt(“7UP”);         // NumberFormatException
double value4 = Double.parseDouble(“3.14”);
```

```
byte  value6 = Byte.parseByte(“1010”, 2); // Decimal value 10.
short value7 = Short.parseShort(“12”, 8); // “012”, not “\012”. Decimal value
10.
int   value8 = Integer.parseInt(“-a”, 16);// Not “-0xa”. Decimal value -10.
long  value9 = Long.parseLong(“-a”, 16);  // Not “-0xa”. Decimal value -10L.
```

Системы представления

```
public class IntegerRepresentation {
  public static void main(String[] args) {
    int positiveInt = +41;     // 0b101001, 051, 0x29
    int negativeInt = -41;     // 0b11111111111111111111111111010111,
-0b101001,
                               // 037777777727, -051, 0xffffffd7, -0x29
    System.out.println("String representation for decimal value: " +
positiveInt);
    integerStringRepresentation(positiveInt);
    System.out.println("String representation for decimal value: " +
negativeInt);
    integerStringRepresentation(negativeInt);
  }

  public static void integerStringRepresentation(int i) {
    System.out.println("    Binary:    " + Integer.toBinaryString(i));
    System.out.println("    Octal:     " + Integer.toOctalString(i));
    System.out.println("    Hex:       " + Integer.toHexString(i));
    System.out.println("    Decimal:   " + Integer.toString(i));

    System.out.println("    Using toString(int i, int base) method:");
    System.out.println("    Base 2:    " + Integer.toString(i, 2));
    System.out.println("    Base 8:    " + Integer.toString(i, 8));
    System.out.println("    Base 16:   " + Integer.toString(i, 16));
    System.out.println("    Base 10:   " + Integer.toString(i, 10));
  }
}
```

Системы представления

```
String representation for decimal value: 41
    Binary:    101001
    Octal:     51
    Hex:       29
    Decimal:   41
    Using toString(int i, int base) method:
    Base 2:    101001
    Base 8:    51
    Base 16:   29
    Base 10:   41
String representation for decimal value: -41
    Binary:    11111111111111111111111111010111
    Octal:     37777777727
    Hex:       ffffffd7
    Decimal:   -41
    Using toString(int i, int base) method:
    Base 2:    -101001
    Base 8:    -51
    Base 16:   -29
    Base 10:   -41
```

Boolean

```
boolean b1 = Boolean.parseBoolean(“TRUE”);        // true.
boolean b2 = Boolean.parseBoolean(“true”);        // true.
boolean b3 = Boolean.parseBoolean(“false”);       // false.
boolean b4 = Boolean.parseBoolean(“FALSE”);       // false.
boolean b5 = Boolean.parseBoolean(“not true”);    // false.
```

Вопросы?

Домашнее задание

Тест

Пожалуйста, пройдите опрос

https://otus.ru/polls/17814/

Спасибо за внимание!

Только хорошего во всех упаковках!