II. Типы, переменные, управляющие инструкции

4. Обёртки примитивных типов

	Значения примитивных типов хранятся непосредственно в переменных и над ними можно выполнять различные операции. Но они не являются объектами. Поэтому у них нет методов, к ним нельзя обращаться используя ссылку типа класса Object и их нельзя хранить в коллекциях.

	Классы обёртки оборачивают примитивные значения и позволяют работать с ними как с объектами. Их можно использовать в коллекциях или как возвращаемое значение функции которая должна возвращать Object. Кроме того классы обёртки предоставляют набор статических функций для работы с примитивными значениями. После создания объекта класса обёртки примитивное значение внутри обёртки не может быть изменено.

Типы обёртки

Примитивный тип	Тип обёртка
byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Все обёртки примитивных типов находятся в пакете java.lang. Этот пакет импортируется по умолчанию поэтому импортировать его не нужно.

Класс Integer

```
public final class Integer extends Number implements Comparable<Integer>
{
   private final int value;

   public Integer(int value)
   public Integer(String s) throws NumberFormatException

   public static Integer valueOf(int i)

   public int intValue(){
       return value;
   }

   public static String toString(int i) {
      if (i == Integer.MIN_VALUE)
         return "-2147483648";
      int size = (i < 0) ? stringSize(-i) + 1 : stringSize(i);
      char[] buf = new char[size];
      getChars(i, size, buf);
      return new String(0, size, buf);
    }

   static void getChars(int i, int index, char[] buf)
   ...
}
```

C	Класс Integer – класс обёртка для int. Оборачиваемое значение хранится в поле value значение которого не может быть изменено. Содержит конструктор для создания объекта класса Integer по значению int и метод для получения значения int хранящегося внутри объекта класса Integer. Содержит статические методы для получения строкового представления для int.

Упаковка и распаковка

	Процесс получения ссылки на объект класса обёртки по значению примитивного типа называется упаковкой, а процесс получения значения примитивного типа по объекту класса обёртки называется распаковкой.

Упаковка

```
public static void main(String[] args) {

      boolean boo = false;
      Boolean wboo = new Boolean(boo);
      System.out.println("boolean variable's value: " + wboo);

      byte b = 2;
      Byte wbyte = new Byte(b);
      System.out.println("byte variable's value: " + wbyte);

      short s = 4;
      Short wshort = new Short(s);
      System.out.println("short variable's value: " + wshort);

      int i = 16;
      Integer wint = new Integer(i);
      System.out.println("int variable's value: " + wint);

      long l = 123;
      Long wlong = new Long(l);
      System.out.println("long variable's value: " + wlong);

      float f = 12.34f;
      Float wfloat = new Float(f);
      System.out.println("float variable's value: " + wfloat);

      double d = 12.56d;
      Double wdouble = new Double(d);
      System.out.println("double variable's value: " + wdouble);
}
```

Упаковка

```
boolean variable's value: false
byte variable's value: 2
short variable's value: 4
int variable's value: 16
long variable's value: 123
float variable's value: 12.34
double variable's value: 12.56
```

Распаковка

```
public class UnwrapDemo {

   public static void main(String[] args) {

      Boolean wboo = new Boolean(false);
      boolean boo  = wboo.booleanValue();
      System.out.println("boolean variable's value: " + boo);

      Byte wbyte = new Byte((byte)2);
      byte b = wbyte.byteValue();
      System.out.println("byte variable's value: " + b);

      Short wshort = new Short((short)4);
      short s = wshort.shortValue();
      System.out.println("short variable's value: " + s);

      Integer wint = new Integer(16);
      int i = wint.intValue();
      System.out.println("int variable's value: " + i);

      Long wlong = new Long(123);
      long l = wlong.longValue();
      System.out.println("long variable's value: " + l);

      Float wfloat = new Float(12.34);
      float f = wfloat.floatValue();
      System.out.println("float variable's value: " + f);

      Double wdouble = new Double(12.56);
      double d = wdouble.doubleValue();
      System.out.println("double variable's value: " + d);
   }
}
```

Распаковка

```
boolean variable's value: false
byte variable's value: 2
short variable's value: 4
int variable's value: 16
long variable's value: 123
float variable's value: 12.34
double variable's value: 12.56
```

Автоматическая упаковка и распаковка

	Автоматическая упаковка и распаковка позволяет не указывать явно функции для упаковки и распаковки. Функции для упаковки и распаковки компилятор добавит автоматически. Автоматическая упаковка происходит при присваивании значения примитивного типа переменной типа класса обёртки. Автоматическая распаковка происходит при присваивании ссылки на объект класса обёртки переменной примитивного типа или при использовании ссылок на объекты классов обёрток в качестве операндов которые должны быть примитивами.

Автоматическая упаковка

```
public class AutoWrapDemo {

   public static void main(String[] args) {

      boolean boo = false;
      Boolean wboo = boo;
      System.out.println("boolean variable's value: " + wboo);

      byte b = 2;
      Byte wbyte = b;
      System.out.println("byte variable's value: " + wbyte);

      short s = 4;
      Short wshort = s;
      System.out.println("short variable's value: " + wshort);

      int i = 16;
      Integer wint = i;
      System.out.println("int variable's value: " + wint);

      long l = 123;
      Long wlong = l;
      System.out.println("long variable's value: " + wlong);

      float f = 12.34f;
      Float wfloat = f;
      System.out.println("float variable's value: " + wfloat);

      double d = 12.56d;
      Double wdouble = d;
      System.out.println("double variable's value: " + wdouble);
     }
}
```

Автоматическая упаковка

```
boolean variable's value: false
byte variable's value: 2
short variable's value: 4
int variable's value: 16
long variable's value: 123
float variable's value: 12.34
double variable's value: 12.56
```

Автоматическая распаковка

```
public class AutoUnwrapDemo {

   public static void main(String[] args) {

      Boolean wboo = new Boolean(false);
      boolean boo  = wboo;
      System.out.println("boolean variable's value: " + boo);

      Byte wbyte = new Byte((byte)2);
      byte b = wbyte;
      System.out.println("byte variable's value: " + b);

      Short wshort = new Short((short)4);
      short s = wshort;
      System.out.println("short variable's value: " + s);

      Integer wint = new Integer(16);
      int i = wint;
      System.out.println("int variable's value: " + i);

      Long wlong = new Long(123);
      long l = wlong;
      System.out.println("long variable's value: " + l);

      Float wfloat = new Float(12.34);
      float f = wfloat;
      System.out.println("float variable's value: " + f);

      Double wdouble = new Double(12.56);
      double d = wdouble;
      System.out.println("double variable's value: " + d);
   }
}
```

Автоматическая распаковка

```
boolean variable's value: false
byte variable's value: 2
short variable's value: 4
int variable's value: 16
long variable's value: 123
float variable's value: 12.34
double variable's value: 12.56
```

Кеширование обёрток

	Неизменность объектов классов обёрток позволяет использовать один объект несколько раз. Для повышения производительности объекты классов обёрток Boolean, … , Integer, Long для некоторого диапазона значений примитивных типов кешируются. Если используется автоматическая упаковка компилятор добавляет инструкции для получения ссылки на объект обёртки из кеша объектов обёрток если необходимый объект обёртка содержится в кеше.

Интересный пример

```
public static void main(String[] args) {

    Integer i = 127;
    Integer j = 127;
    System.out.println("i==j >> " + (i == j));

    Integer a = 127;
    Integer b = new Integer(127);
    System.out.println("a==b >> " + (a == b));

    Integer k = 128;
    Integer l = 128;
    System.out.println("k==l >> " + (k == l));
}
```

C	Проводится упаковка примитивных типов в объекты. Используется два вида упаковки автоматическая и не автоматическая. Ссылки на объекты проверяются на равенство.

Неожиданный результат

```
i==j >> true
a==b >> false
k==l >> false
```

	Получается что ссылка на обёртку 127 равна ссылке на обёртку 127 в первом случае но не равна во втором, а ссылка на обёртку 128 не равна ссылке на обёртку 128.

Класс Integer

```
public final class Integer extends Number implements Comparable<Integer>
{
   private final int value;

   public Integer(int value)

   private static class IntegerCache {
      static final int high;
      static final Integer cache[];

      static {
         final int low = -128;

         int h = 127;
         ... // high value may be configured by property
         high = h;

         cache = new Integer[(high - low) + 1];
         int j = low;
         for(int k = 0; k < cache.length; k++)
            cache[k] = new Integer(j++);
        }

        private IntegerCache() {}
    }

   public static Integer valueOf(int i) {
      if (i >= -128 && i <= IntegerCache.high)
         return IntegerCache.cache[i + 128];
      else
         return new Integer(i);
   }
   ...
}
```

Класс Integer содержит кеш объектов обёрток для значений int от -128 до 127. При автоматической упаковке компилятор добавляет вызов valueOf. Если нужный объект обёртки есть в кеше возвращается ссылка на него, если нет возвращается ссылка на объект создаваемый с помощью конструктора. При этом новый объект обёртки в кеш не добавляется.

Ошибки с автоматической упаковкой / распаковкой

Медленный цикл

```
public class SlowCycleDemo {

   public static void main(String[] args) {

      long time = System.currentTimeMillis();

      for (Long i = 0L; i <= 100000000L; i++) {

      }

      time = System.currentTimeMillis() - time;

      System.out.println("It took " + time + " milliseconds to execute this cycle");
   }
}
```

C	Замеряется время выполнения 100000000 итераций пустого цикла. В качестве счётчика итераций цикла используется переменная типа Long т.е. типа класса обёртки. Это значит что увеличение счётчика потребует распаковку и упаковку. Затрат на упаковку и распаковку можно избежать если в качестве счётчика использовать переменную примитивного типа long.

Медленный цикл

```
It took 1375 milliseconds to execute this cycle
```

```
It took 156 milliseconds to execute this cycle
```

	Показан вывод для цикла который использует в качестве счётчика Long и для цикла который использует в качестве счётчика long. Использование long почти на порядок быстрее, так как не требует затрат на распаковку и упаковку.

Ещё один медленный цикл

```
public class AnotherSlowCycleDemo {

   public static void main(String[] args) {

      long time = System.currentTimeMillis();

      Long sum = 0L;

      for (long i = 0; i <= 100000000L; i++) {

         sum += i;
      }

      time = System.currentTimeMillis() - time;
      System.out.println(time);
      System.out.println("Sum is: " + sum);
   }
}
```

C	Замеряется время выполнения 100000000 итераций цикла в котором происходит накопление значений в переменной типа Long т.е. типа класса обёртки. Это значит что операция += потребует распаковку и упаковку. Затрат на упаковку и распаковку можно избежать если использовать переменную примитивного типа long.

Ещё один медленный цикл

```
It took 1110 milliseconds to execute this cycle
Sum is: 5000000050000000
```

```
It took 203 milliseconds to execute this cycle
Sum is: 5000000050000000
```

	Показан вывод для цикла который использует для накопления значений переменную типа Long и для цикла который использует для накопления значений переменную типа long. Использование long в 5-6 раз быстрее, так как не требует затрат на распаковку и упаковку.

Ошибка с оператором ==

```
public class WrapErrorDemo {

   public static void main(String[] args) {

      System.out.println("Compare 50 and 100 : " + compare(50, 100));
      System.out.println("Compare 100 and 50 : " + compare(100, 50));
      System.out.println("Compare 50 and 50 : " + compare(50, 50));

      System.out.println("Compare 200 and 200 : " + compare(200, 200));
   }

   public static int compare(Integer first, Integer second) {

      return first < second ? -1 : (first == second ? 0 : 1);
   }
}
```

C	При передаче значений примитивных типов в качестве параметров в метод compare происходит автоматическая упаковка. Оператор > приводит к автоматической распаковке и сравнению примитивных значений. Оператор == может применяться к ссылкам и поэтому не приводит к распаковке и сравнению значений примитивов. Вместо этого он сравнивает ссылки которые будут равны только для кешированных объектов обёрток.

Ошибка с оператором ==

```
Compare 50 and 100 : -1
Compare 100 and 50 : 1
Compare 50 and 50 : 0
Compare 200 and 200 : 1
```

	Два различных объекта обёртки для значения 200 располагаются в различных областях памяти поэтому оператор == вернёт false и получится что 200 > 200.