**Лабораторная работа № 5.**

**Работа с классами-обертками.**

У примитивов в программировании, и в Java в частности, есть множество преимуществ: они занимают мало памяти, за счет чего повышается эффективность работы программы, и четко разделены по диапазонам значений. Однако в процессе изучения Java мы уже не раз, словно мантру, повторяли — “**в Java все является объектом**”. А ведь примитивы — прямое опровержение этих слов. Объектами они не являются. Получается, принцип “все является объектом” является ложным? На самом деле нет. В Java у каждого примитивного типа есть свой брат-близнец — **класс-обертка** (**Wrapper**). Что такое обертка? **Обертка — это специальный класс, который хранит внутри себя значение примитива.** Но поскольку это именно класс, он может создавать свои экземпляры. Они будут хранить внутри нужные значения примитивов, при этом будут являться настоящими объектами. Названия классов-оберток очень похожи на названия соответствующих примитивов, или полностью с ними совпадают. Поэтому запомнить их будет очень легко.

|  |  |
| --- | --- |
| **Primitive Data Types** | **Wrapper Classes** |
| int | Integer |
| short | Short |
| long | Long |
| byte | Byte |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

Объекты классов оберток создаются так же, как и любые другие:

public static void main(String[] args) {

Integer i = new Integer(682);

Double d = new Double(2.33);

Boolean b = new Boolean(false);

}

Классы-обертки позволяют нивелировать недостатки, которые есть у примитивных типов. Самый очевидный из них — **примитивы не имеют методов**. Например, у них нет метода toString(), поэтому ты не сможешь, например, преобразовать число int в строку. А вот с классом-оберткой Integer — запросто.

public static void main(String[] args) {

Integer i = new Integer(432);

String s = i.toString();

}

Возникнут сложности и с обратным преобразованием. Допустим, у нас есть строка, про которую мы точно знаем, что она содержит число. Тем не менее, в случае с примитивным типом int мы никак не сможем это число из строки достать и превратить, собственно, в число. Но благодаря классам-оберткам такая возможность у нас появилась.

public static void main(String[] args) {

String s = "1166628";

Integer i = Integer.parseInt(s);

System.out.println(i);

}

**Вывод: 1166628** Мы успешно получили число из строки и присвоили его в переменную-ссылку Integer i. Кстати, по поводу ссылок. Ты уже знаешь, что параметры передаются в методы по-разному: примитивы — по значению, а объекты — по ссылке. Ты можешь использовать это знание при создании своих методов: если твой метод работает, например, с дробными числами, но тебе нужна логика именно передачи по ссылке, ты можешь передать в метод параметры Double/Float вместо double/float. Кроме того, помимо методов в классах-обертках есть очень удобные для использования статические поля. Например, представь, что перед тобой сейчас стоит задача: вывести в консоль максимально возможное число int, а после — минимально возможное. Классы-обертки легко позволяют решать такие “бытовые задачи”:

public class Main {

public static void main(String[] args) {

System.out.println(Integer.MAX\_VALUE);

System.out.println(Integer.MIN\_VALUE);

}

}

Такие поля позволяют не отвлекаться от выполнения более серьезных задач. Не говоря уж о том, что в процессе печати числа 2147483647 (это как раз MAX\_VALUE) не мудрено и опечататься:) Кроме того, в одной из прошлых лекций мы уже обращали внимание на то, что **объекты классов-оберток являются неизменяемыми (Immutable)**.

public static void main(String[] args) {

Integer a = new Integer(0);

Integer b = new Integer(0);

b = a;

a = 1;

System.out.println(b);

}

**Вывод: 0** Объект, на который изначально указывала ссылка а, не изменил свое состояние, иначе значение b тоже изменилось бы. Как и в случае со String, вместо изменения состояния объекта-обертки в памяти создается абсолютно новый объект. Почему же создатели Java, в конечном итоге, приняли решение оставить в языке примитивные типы? Раз уж все должно являться объектом, и у нас уже есть классы-обертки, которыми можно выразить все, что выражают примитивы, почему вообще не оставить в языке только их, а примитивы удалить? Ответ прост — производительность. Примитивные типы потому и называют примитивными, потому что они лишены многих “тяжеловесных” особенностей объектов. Да, у объекта есть много удобных методов, но ведь они не всегда тебе нужны. Иногда тебе нужно просто число 33, или 2,62, или значение true/false. В ситуациях, когда все преимущества объектов не имеют значения и не нужны для работы программы, примитивы справятся с задачей гораздо лучше.

**Автоупаковка/автораспаковка**

Одной из особенностей примитивов и их классов-оберток в Java является автоупаковка/автораспаковка (Autoboxing/Autounboxing). Как мы с тобой уже узнали ранее, Java — объектно-ориентированный язык. Это значит, что все программы, написанные на Java, состоят из объектов. Примитивы не являются объектами. Но при этом переменной класса-обертки можно присваивать значение примитивного типа. Этот процесс называется **автоупаковкой** (**autoboxing**). Точно так же переменной примитивного типа можно присваивать объект класса-обертки. **Этот процесс называется автораспаковкой (autounboxing)**. Например:

public class Main {

public static void main(String[] args) {

int x = 7;

Integer y = 111;

x = y; // автораспаковка

y = x \* 123; // автоупаковка

}

}

В строке 5 мы присваиваем примитиву x значение y, который является объектом класса-обертки Integer. Как видишь, никаких дополнительных действий для этого не нужно: **компилятор знает, что** int **и** Integer**, по сути, одно и то же**. Это и есть автораспаковка. Так же происходит и автоупаковка в строке 6: объекту y легко присваивается значение примитивов (х\*123). Это пример автоупаковки. Именно поэтому добавляется слово "авто": **для присваивания ссылок-примитивов объектам их классов-оберток (и наоборот) не требуется ничего делать, все происходит автоматически**. Удобно, да? :) Еще одно очень большое удобство автоупаковки/автораспаковки проявляется в работе методов. Дело в том, что **параметры методов тоже подлежат автоупаковке и автораспаковке**. И, например, если какой-то из них методов принимает на вход два объекта Integer — мы легко можем передать туда обычные примитивы int!

public class Main {

public static void main(String[] args) {

printNumber(7);//обычный int, даже без переменной

}

public static void printNumber(Integer i) {

System.out.println("Вы ввели число " + i);

}

}

**Вывод: Вы ввели число 7**

Точно так же работает и наоборот:

public class Main {

public static void main(String[] args) {

printNumber(new Integer(632));

}

public static void printNumber(int i) {

System.out.println("Вы ввели число " + i);

}

}

Важный момент, о котором нужно помнить: **автоупаковка и распаковка не работают для массивов**!

public class Main {

public static void main(String[] args) {

int[] i = {1,2,3,4,5};

printArray(i);//ошибка, не компилируется!

}

public static void printArray(Integer[] arr) {

System.out.println(Arrays.toString(arr));

}

}

Попытка передать массив примитивов в метод, который принимает на вход массив объектов, вызовет ошибку компиляции. Напоследок, еще раз кратко сравним примитивы и обертки.

**Примитивы:**

* Имеют преимущество в производительности.

**Обертки:**

* Позволяют не нарушать принцип “все является объектом”, благодаря чему числа, символы и булевы значения true/false не выпадают из этой концепции.
* Расширяют возможности работы с этими значениями, предоставляя удобные методы и поля.
* Необходимы, когда какой-то метод может работать исключительно с объектами.

**Задание**

Реализовать калькулятор, поддерживающий функции калькулятора Windows, с использованием классов-оберток.