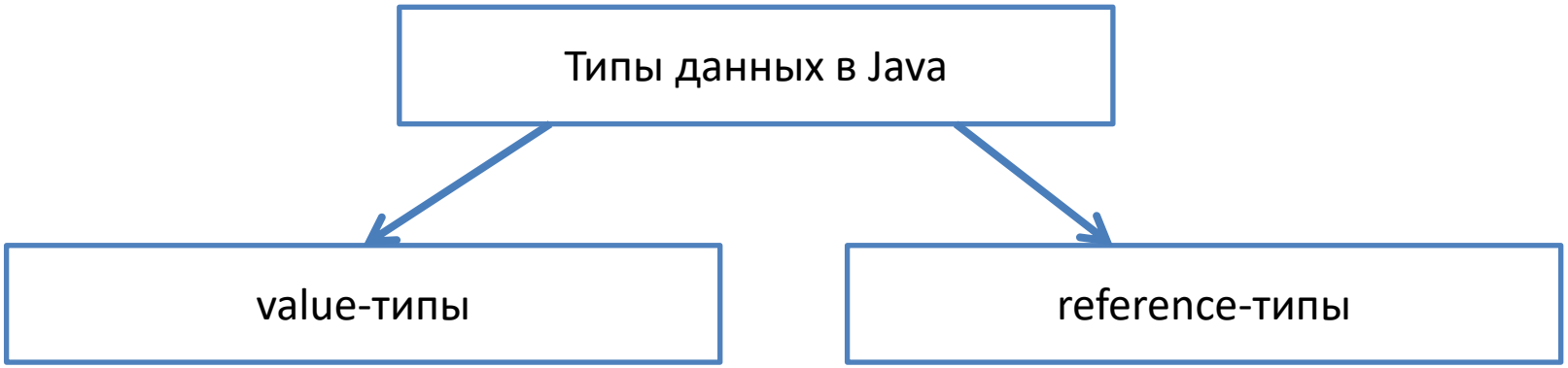
Лекция 8. Value-типы и ссылочные типы. Символьный тип

Типы данных в Java

- Все типы в Java можно разделить на две категории: **value-типы (типы значений)** и **reference-типы (ссылочные типы)**
- Типы из данных категорий ведут себя по-разному
- К **value-типам** относятся, например числа, а к **reference-типам** относятся строки

Типы данных в Java

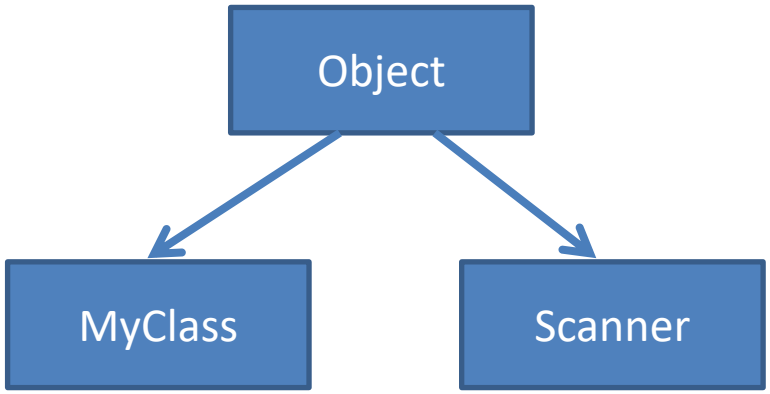


Числовые целые:
byte, short, int, long

Вещественные:
float, double

Логический: boolean

Символьный: char



Все классы наследуются от класса Object

Value-типы

- Их 8 штук
- Все числовые типы:
  byte, short, int, long; float, double
- Логический тип boolean
- Символьный тип char (рассмотрим позже)

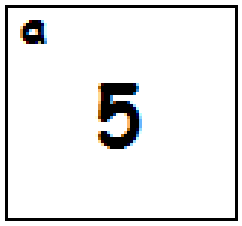
- В Java эти типы еще называют **примитивными**, т.к. они встроены в язык и не являются объектами

Value-типы

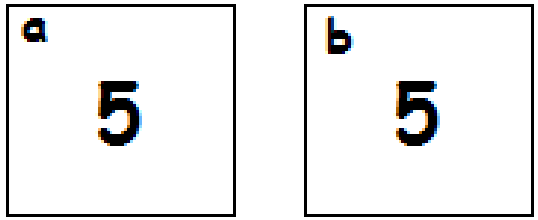
- Переменные **value-типов** хранят само значение типа
- При присваивании происходит копирование значения
- При передаче аргументов в функции, происходит копирование аргумента

Как работают value-типы

- int a = 5;



- int b = a;



- Если изменить **a** или **b**, то это не повлияет на другую переменную

Reference-типы

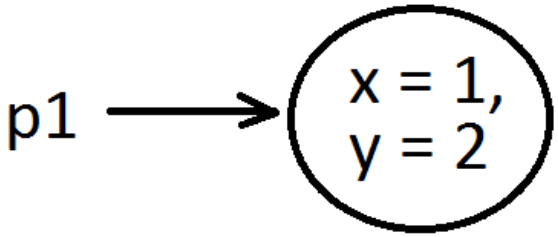
- Переменные **reference-типов** хранят не само значение, а **ссылку** на него
- Point p1 = new Point(1, 2);
- Здесь **p1** хранит не сам объект Point, а только **ссылку** на него
- А сам объект хранится отдельно в области оперативной памяти, называемой **кучей (heap)**
- **Куча (heap)** – это область памяти, выделенная ОС для использования приложением
- Все объекты создаются и хранятся именно в **куче**

Reference-типы

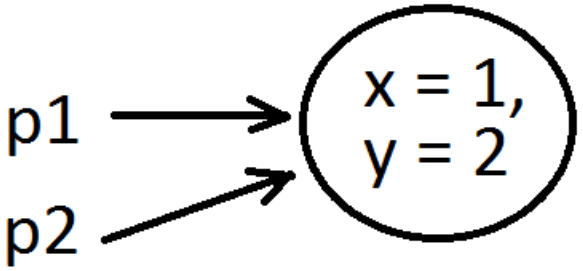
- **Ссылка (reference)** – это некоторая сущность, указывающая на данные, но не хранящая их
- По сути ссылка хранит адрес в памяти (в куче), начиная с которого начинается объект
- Этот адрес является целым числом
- Но у нас нет возможности работать со ссылкой как с числом – это от нас скрыто

Reference-типы

- Переменная **p1** хранит ссылку на объект:
- Point p1 = new Point(1, 2);



- При присваивании происходит копирование ссылки:
- Point p2 = p1;

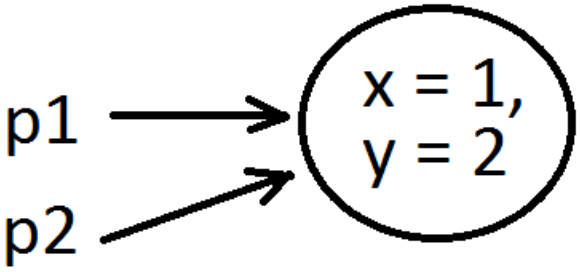


- Т.е. получается, что к одному и тому же объекту можно обращаться по двум именам - **p1** и **p2**

- В Java все классы являются **reference-типами**

Reference-типы

- Т.к. эти ссылки указывают на один и тот же объект, то мы получим измененное значение:



- p2.setX(3);
  System.out.println(p1.getX()); // 3

- При передаче объекта в функцию, происходит копирование ссылки на него
- Аналогично при возврате объекта из функции

Зачем нужны ссылки?

- Объекты часто являются большими, и копировать их очень затратно по времени и памяти
- При использовании ссылок копируется только ссылка, которая по сути является целым числом. Это быстро и требует мало памяти

Указатели

- В некоторых языках, например, в C и C++, есть понятие **указатель (pointer)**
- **Указатель** – это переменная, которая может хранить адрес в памяти (целое число)
- В Java в языке нет **указателей**, вместо них используются **ссылки**
- С указателями, в отличие от ссылок, можно работать как с числом
- Например, если к указателю прибавить 4, то мы сдвинемся на 4 байта и т.д.

Проверка объектов на равенство

- Для объектов нельзя использовать проверку через == и != для проверки равенства
- Для объектов оператор == проверяет, что ссылки указывают на один и тот же объект в памяти или нет
- Аналогично != проверяет, что ссылки указывают на разные объекты
- Чтобы сравнить содержимое объектов, нужно использовать метод **equals**
- boolean x = o1.equals(o2);

Логика equals по умолчанию

- Чтобы **equals** работал для наших классов, нам нужно самим реализовать этот метод
- А по умолчанию **equals** работает так же, как сравнение ссылок
- Переопределение **equals** будет рассмотрено на курсе ООП
- У многих стандартных классов, например, у строк, **equals** уже реализован правильно

Значение null

Значение null

- Переменные ссылочных типов могут принимать специальное значение null
- Пример: String s = null;
- Оно означает пустую ссылку, то есть адрес, который не указывает ни на какой объект

Падение кода при обращении к null

- Если вызвать функцию (или обратиться к полю) для переменной, которая имеет значение null, то произойдет ошибка NullPointerException во время работы программы
- String s = null;
  int length = s.length(); // код упадет
- Код падает, т.к. объекта нет – здесь не у кого вызывать функцию **length()**

Для чего используется null?

- Значение null используется, чтобы показать отсутствие данных
- Например, у нас есть некоторый объект Person, у которого есть поле типа Cat
- Но у некоторого человека может не быть кошки, тогда значение поля у него будет null

Для чего используется null?

- Также значение null может быть полезно, если мы хотим показать, что функция отработала, но получить результат не удалось
- Например, мы написали функцию, которая ищет строку нужной длины среди некоторого набора строк
- Но такой строки не оказалось
- В этом случае функция может вернуть null, а вызывающий код проверить, что результат равен null и, например, напечатать сообщение, что ничего не найдено

Для чего используется null?

```
public static String findString(int length) {
    // код, который делает return, если нашел строку

    // в конце делается return null если ничего не найдено
    return null;
}

public static void main(String[] args) {
    String result = findString(4);
    if (result == null) {
        System.out.println(“Ничего не найдено”);
    } else {
        System.out.println(“Результат = ” + result);
    }
}
```

Проверка на null

- Чтобы код не падал, нужно обязательно проверять на null в тех местах, где он может быть
-
```
public static void main(String[] args) {
    String result = findString(4);

    if (result == null) {
        System.out.println(“Ничего не найдено”);
    } else {
        System.out.println(“Результат = ” + result);
    }
}
```
- Проверять на null нужно через == и !=
- Код через **equals** может падать, если вызывать его от null

Символьный тип char

Символьный тип char

- Кроме строкового типа, в Java есть символьный тип char
- Это примитивный тип
- Размер переменной – 2 байта
- Его переменные могут хранить один символ
- Литералы заключены в одинарные кавычки: ‘a’, ‘5’, ‘\\’, ‘\n’
- char lineSeparator = ‘\n’;

Символьный тип char

- У строк можно брать символ по порядковому номеру (отсчитывается от 0)
- String s = “ABCDE”;
- char secondSymbol = s.charAt(1); // B
- char lastSymbol = s.charAt(5);
  // ошибка при исполнении программы –
  // выход за границы строки
- Правильно:
  char lastSymbol = s.charAt(s.length() - 1);

Функции для работы с символами

- Стандартный класс Character:
  - boolean isDigit(char c) – проверка, что цифра
  - boolean isLetter(char c) – проверка, что буква
  - boolean isLetterOrDigit(char c) – что буква или цифра
  - boolean isLowerCase(char c) – что буква в нижнем регистре
  - boolean isUpperCase(char c) – что буква в верхнем регистре
- Пример:

```
boolean isDigit = Character.isDigit('4');   // true
```

Пробельные символы

- boolean Character.isWhitespace(char c) – проверка, что пробельный символ
- **Пробельными символами** считаются пробел, табуляция и перевод строки

Функции работы с символами

- Стандартный класс Character:
  - char toUpperCase(char c) – перевод в верхний регистр
  - char toLowerCase(char c) – перевод в нижний регистр

- Если символ уже в этом регистре, или не буква, то выдается сам символ
- **Пример:**
- ```
  char lowerCaseChar1 = Character.toLowerCase('A');   // a
  char lowerCaseChar2 = Character.toLowerCase('a');   // a
  char upperCaseChar1 = Character.toUpperCase('A');   // A
  char upperCaseChar2 = Character.toUpperCase('a');   // A
  ```
  ```

Пример работы со строками

```
Scanner scanner = new Scanner(System.in);
String name = scanner.nextLine();
if (Character.isLowerCase(name.charAt(0))) {
    System.out.println(
      “Имя должно начинаться с заглавной буквы!”);
    return;
}
```

Проход по всем символам строки

- ```
  Scanner scanner = new Scanner(System.in);
  String s = scanner.nextLine();

  for (int i = 0; i < s.length(); ++i) {
      char c = s.charAt(i);
      // работаем с текущим символом c
  }
  ```
  ```

Задача на дом «Подсчет символов»

- Прочитать с консоли строку
- Вывести число букв в этой строке
- Вывести число цифр в этой строке
- Вывести число пробельных символов в этой строке
- Вывести число остальных символов в строке

Задача на курс «Макс. подстрока»

- Написать функцию, которая ищет в строке подстроку максимальной длины, состоящую из одного и того же символа, и выдает эту максимальную длину
- Например, есть строка "аааббдеггггв", должно выдаться число 4, потому что есть 4 подряд символа «г», и это максимальная подстрока, где подряд идет один и тот же символ
- Функция должна работать без учета регистра

Задача на курс «Палиндром»

- Объявить некоторую строковую переменную в программе
- Проверить, что данная строка является палиндромом – то есть читается одинаково слева направо и справа налево.
- При проверке не учитывать регистр символов, учитывать только буквы
- Пример палиндрома: «Аргентина манит негра»

- **Требование**: сделать без создания новой строки и без удаления символов из строки