Третья лекция
Логика, Циклы

Study JAVA

Основные понятия объектно-ориентированного программирования

Основными концепциями здесь являются понятия объект и класс.

3 главных принципа, на которых строится объектно-ориентированное программирование:

Инкапсуляция;
Наследование;
Полиморфизм.

Инкапсуляция означает, что каждый объект имеет свои копии переменных. Данные объекта недоступны его клиентам непосредственно. Вместо этого они **инкапсулируются — скрываются** от прямого доступа извне. То есть данные объекта можно изменить только посредством методов самого объекта.

Инкапсуляция реализуется с помощью модификаторов доступа

Методы, классы и атрибуты могут иметь следующую область видимости:
public —доступ без каких-либо ограничений;
private — доступ разрешен только из данного класса;
protected — доступ разрешен из данного класса и из всех классов-потомков, а также из всех классов данного пакета.
„default"— доступ разрешен из всех классов данного пакета

Модификатор доступа	Тот же самый класс	Тот же самый пакет	Субкласс (класс потомок, возможно из другого пакета)	Внешние классы из других пакетов
`private`	Да			
модификатор доступа опущен – пакетный доступ (default)	Да	Да		
`protected`	Да	Да	Да	
`public`	Да	Да	Да	Да

Study JAVA

Ключевые слова Java

Ключевые слова нельзя использовать для именования конструкций Java: переменных, методов и т.д.

abstract	assert ***	boolean	break	byte
case	catch	char	class	const *
continue	default	do	double	else
enum ****	extends	final	finally	float
for	goto *	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp **	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

Пример использования get и set

```
public class User {
    private int age;//поле с модификатором private

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

Арифметические операторы

+	a + b	Сложение a и b
-	a – b	Вычитание b из a
-	-a	Арифметическая инверсия a
*	a * b	Умножение a на b
/	a / b	Деление a на b
%	a % b	Вычисление остатка от деления a на b (деление по модулю)
+=	a += b	a = a + b (сложение с присваиванием)
-=	a -= b	a = a – b (вычитание с присваиванием)
*=	a *= b	a = a * b (умножение с присваиванием)
/=	a /= b	a = a / b (деление с присваиванием)
%=	a %= b	a = a % b (деление по модулю с присваиванием)

Study JAVA

Инкремент и декремент

Операторы инкремента(++) и декремента(--) это операторы, которые увеличивают или уменьшают на единицу значение переменной при использовании инкремента и декремента соответственно

Различают префиксную и постфиксную форму для обоих операторов. Префиксная форма сначала изменяет значение переменной на 1 и использует новое значение для дальнейших вычислений. Постфиксная форма сначала использует старое значение операнда, а затем изменяет его на 1

Префиксный оператор инкремента:
int i = 1;
int m = 2 * ++i; // результат m =
	Постфиксный оператор инкремента:
	int i = 1;
	int m = 2 * i++; // результат m = 2

Префиксный оператор декремента:
int i = 2;
int m = 2 * --i; // результат m =
	Постфиксный оператор декремента:
	int i = 2;
	int m = 2 * i--; // результат m = 4

Приведение типов

Иногда возникают ситуации, когда имея величину какого-либо определенного типа, необходимо присвоить ее переменной другого типа.
Ручное приведение типов или явное приведение – с использованием конструкции (<тип>)

```
double a = 56.9898;
int b = (int)a;// при этом дробная часть отбросится
```

Автоматическое преобразование типов данных происходит в том случае, если мы присваиваем одной переменной значение переменной другого типа, и при этом выполняются следующие условия:

- типы данных должны быть совместимы (например, нельзя присваивать переменной типа boolean значение типа int);
- тип данных переменной, которой присваивается значение, должен быть больше по размеру типа данных переменной, чье значение присваивается. Например, можно преобразовывать значение типа short в тип int. Это называется расширяющим преобразованием.

При делении целых чисел друг на друга преобразование к double или float не производится – и дробная часть отбрасывается. Поэтому, если она необходима – нужно преобразовать один и операндов в double вручную

```
double d = 3 / 2; // d будет иметь значение 1.0
d = (double) 3 / 2; // d будет иметь значение 1.5
```

Логические операторы в Java.

В логическом выражении могут использоваться

следующие логические операторы:

= = Равно
!= Не равно
< Меньше
< = Меньше или равно
> Больше
> = Больше или равно
& Логическое и
| Логическое или
! Отрицание
&& Условное и
| | Условное или

*Операторы & и | всегда проверяют значение обоих операндов.

Study JAVA

Логические и условные операторы

Для того, чтобы изменить последовательность выполнения фрагментов программы, в языке Java применяются конструкции if, if-else и switch. Конструкция if-else имеет следующий вид:

```
if (логическое выражение) {
    //блок кода 1
}
else{
    //блок кода 2. Блок else не является обязательным.
}
```

Логическое выражение это выражение (или переменная), возвращающее значение типа boolean.

```
boolean b = true;
if (b) {
 System.out.println("b - истина");
}
```

Условный оператор if

Ели логическое выражение в скобках правдиво, то выполняется , блок кода в фигурных скобках {} после if. Если логическое выражение принимает значение false, то ничего не происходит.

```
if (a == b) {
    //Если a равно b - выводим сообщение
    System.out.println("a и b равны!");
}
```

Условный оператор if-else

Конструкция if-else отличается от предыдущей тем, что если логическое выражение в круглых скобках принимает значение false, то выполняется блок кода, находящийся в фигурных скобках после ключевого слова else

```
if (a == b) {
    //Если a равно b - выводим сообщение
    System.out.println("a и b равны!");
}
else{
    //Если a не равно b - выводим сообщение
    System.out.println("a и b не равны!");
}
```

Также можно создавать составные операторы if содержащие несколько **else if блоков**.

```
public char convertGrades( int testResult){

char grade;

    if (testResult >= 90){
        grade = 'A';
    }else if (testResult >= 80 && testResult < 90){
        grade = 'B';
    }else if (testResult >= 70 && testResult < 80){
        grade = 'C';
    }else {
        grade = 'D'; //Всегда заканчивается else
    }
        return grade;
}
```

Study JAVA

Возможные сокращения if-else

1. Если блоки кода if, else содержат лишь одну строку, то можно обойтись без использования фигурных скобок. Предыдущий пример можем записать так:

```
if (a == b)
    System.out.println("a и b равны!");
else
    System.out.println("a и b не равны!");
```

2. Существует еще один способ сокращения оператора if-else при помощи оператора **? :** . Запишем предыдущий пример следующим образом:

```
System.out.println(a==b ? "a и b равны!" : "a и b не равны!");
```

Общая форма записи условия выглядит следующим образом:

Логическое_выражение ? Выражение1 : Выражение2

Если Логическое_выражение принимает значение true, то выполняется Выражение1, а иначе — Выражение2.
Внимание: при использовании этой конструкции два последних выражения должны иметь один и тот же тип.

Условный оператор switch — case удобен в тех случаях, когда количество вариантов очень много и писать для каждого if-else очень долго.

Конструкция имеет следующий вид :

```
switch (выражение) {
  case значение1:
    //блок кода 1;
    break;
  case значение2:
    //блок кода 2;
    break;
    ...
  case значениеN:
    //блок кода N;
    break;
  default:
    блок N+1;
}
```

Выражение в круглых скобках после switch сравнивается со значениями, указанными после слова case, в случае совпадения, управление передается соответствующему блоку кода. Если выражение не совпадает ни с одним вариантом case, то управление передается блоку default, который не является обязательным. После выполнения соответствующего блока, оператор break вызывает завершение выполнения оператора switch. Если break отсутствует, то управление передается следующему блоку за только что выполненным.

Study JAVA

Пример switch - case

```
int day = 3;
    String dayString;
    switch (day) {
        case 1:  dayString = "Понедельник";
                 break;
        case 2:  dayString = "Вторник";
                 break;
        case 3:  dayString = "Среда";
                 break;
        case 4:  dayString = "Четверг";
                 break;
        case 5:  dayString = "Пятница";
                 break;
        case 6:  dayString = "Суббота";
                 break;
        case 7:  dayString = "Воскресенье";
                 break;
        default: dayString = "Ошибка";
                 break;
    }
    System.out.print(dayString);
```

Циклы

Цикл в программировании используется для многократного повторения определенного фрагмента кода.

В Java существует 3 оператора цикла:

- **for**
- **while**
- **do-while.**

Конструкция for

Конструкция for имеет следующий вид:

```
for (int i = 1; i <= 10; i++) {
 System.out.print(i + " ");
}
```

```
for (инициализация; условие; итерация) {
    //тело цикла
}
```

Инициализация — первый параметр, который содержит переменную и ее начальное значение. С помощью этой переменной будет подсчитываться количество повторений цикла. В нашем примере это переменная int i = 1.

Условие — второй параметр, содержит некоторое логическое выражение — условие при котором будет выполняться цикл. В нашем примере это условие i <= 10.

Итерация — третий параметр, выражение, изменяющее переменную (заданную в инициализации) после каждого шага цикла. Изменение происходит только в случае выполнения условия. В нашем примере итерация i++ — увеличение переменной на единицу. Также для итерации часто используется i— — уменьшение переменной на единицу.

Цикл while в Java.

Пример: Вывод на экран значений от 1 до 10.

```
int i = 1;
    while(i < 11){
     System.out.println("i= " + i);
           i++;
}
```

Конструкция while имеет следующий вид:

```
while(логическое_выражение) {
   //тело цикла
}
```

Для реализации бесконечного цикла, в качестве параметра достаточно указать true

```
while(true) {

//тело цикла

}
```

Или приведем пример цикла, который не выполнится ни разу:

```
int i =10;
while(i < 5){
    System.out.println("i= " + i);
    i++;
}
```

Цикл do-while в Java.

В отличие от оператора while, do-while является оператором постусловия, который сначала выполняет тело цикла, а потом осуществляет проверку условия. Таким образом, тело цикла выполнится всегда хотя бы один раз.

Пример: Вывод на экран значений от 1 до 10.

```
int i = 1;

do{

    System.out.println("i = " + i);
   i++;

}while(i < 11);
```

Конструкция **do — while** имеет следующий вид:

```
do {
   //тело цикла
}while(логическое_выражение);
```

Досрочный выход из цикла break

Выполнение цикла можно прервать, если внутри тела цикла вызвать оператор break. После выполнения оператора **break** произойдет моментальный выход из цикла (не будет окончена даже текущая итерация). Управление передастся оператору, следующему за циклом.

```
int i =0;
while (i< 100){
      if (i == 3){
      break; // Выпрыгиваем из цикла
     }
       System.out.println("i =  "+ i);
       i++;
 }
```

Оператор continue

В некоторых ситуациях возникает потребность досрочно перейти к выполнению следующей итерации, проигнорировав часть операторов тела цикла, еще не выполненных в текущей итерации. В этом случае необходимо сделать переход из текущей точки в конец цикла, т.е. в точку после последнего оператора тела цикла.

```
for (int i = 0; i < 10; i++) {

        if (i % 2 == 0) continue;

        System.out.print(i + " ");
    }
```

Study JAVA

Java класс Math

Разработчику на Java доступно множество готовых (или библиотечных) классов и методов, полезных для использования в собственных программах. Наличие библиотечных решений позволяет изящно решать множество типовых задач.

Math.abs(n) — возвращает модуль числа n.
Math.round(n) — возвращает целое число, ближайшее к вещественному числу n (округляет n).
Math.ceil(n) — возвращает ближайшее к числу n справа число с нулевой дробной частью (например, Math.ceil(3.4) в результате вернёт 4.0).
Math.cos(n), Math.sin(n), Math.tan(n) — тригонометрические функции sin, cos и tg от аргумента n, указанного в радианах.
Math.acos(n), Math.asin(n), Math.atan(n) — обратные тригонометрические функции, возвращают угол в радианах.
Math.toDegrees(n) — возвращает градусную меру угла в n радианов.
Math.toRadians(n) — возвращает радианную меру угла в n градусов.
Math.sqrt(n) — возвращает квадратный корень из n.
Math.pow(n, b) — возвращает значение степенной функции n в степени b, основание и показатель степени могут быть вещественными.
Math.log(n) — возвращает значение натурального логарифма числа n.
Math.log10(n) — возвращает значение десятичного логарифма числа n.

Все перечисленные функции принимают вещественные аргументы, а тип возвращаемого значения зависит от типа аргумента и от самой функции.

Кроме функций в рассматриваемом классе имеются две часто используемых константы:

Math.PI — число «пи», с точностью в 15 десятичных знаков.
Math.E — число Неппера (основание экспоненциальной функции), с точностью в 15 десятичных знаков.

Примеры использования:

```
System.out.println(Math.abs(-2.33)); // выведет 2.33
System.out.println(Math.round(Math.PI)); // выведет 3
System.out.println(Math.round(9.5)); // выведет 10
System.out.println(Math.round(9.5-0.001)); // выведет 9
System.out.println(Math.ceil(9.4)); // выведет 10.0
double c = Math.sqrt(3*3 + 4*4);
System.out.println(c); // выведет гипотенузу треугольника с катетами 3 и 4
double s1 = Math.cos(Math.toRadians(60));
System.out.println(s1); // выведет косинус угла в 60 градусов
```

теоретически это все-таки возможно, именно поэтому генерируемые функцией числа называются не случайными, а псевдослучайными).

Псевдослучайные числа

Math.random() возвращает псевдослучайное вещественное число из промежутка [0;1).

Если требуется получить число из другого диапазона, то значение функции можно умножать на что-то, сдвигать и, при необходимости, приводить к целым числам.

Study JAVA

Примеры:

```
System.out.println(Math.random()); // вещественное число из [0;1)
System.out.println(Math.random()+3); // вещественное число из [3;4)
System.out.println(Math.random()*5); // вещественное число из [0;5)
System.out.println( (int)(Math.random()*5) ); // целое число из [0;4]
System.out.println(Math.random()*5+3); // вещественное число из [3;8)
System.out.println( (int)(Math.random()*5+3) ); // целое число из [3;7]
System.out.println( (int)(Math.random()*5)-1 ); // целое число из [-1;3]
```

Рассмот… для того, чтобы из… ное число из отрез…