



# Объектно-ориентированные принципы Java

## Лекция 3

# Цели



- Знакомство с классами и объектами.
- Описать концепцию методов в классе.
- Знакомство с конструкторами.
- Описать принципы наследования, полиморфизма, инкапсуляции.
- Объяснить различие между перегрузкой и замещением методов.
- Идентифицировать спецификаторы доступа и модификаторы методов.

# Классы и объекты

- **Java** – это объектно-ориентированный язык программирования.
- В Java все является объектом.
- Объекты создаются на базе встроенных или определяемых пользователем классов.
- Под **объектом** подразумевается некоторая сущность, обладающая состоянием и поведением.
- Как правило, при рассмотрении объектов выделяется, что объекты принадлежат одному или нескольким классам, которые в свою очередь определяют поведение объекта.

# Классы и объекты

- **Класс** - абстрактный тип данных.
- **Класс** - коллекция методов и данных.
- Данные и методы, обернутые в единое целое, называются **классом**.
- **Класс** – основной стандартный блок объектно-ориентированного языка типа Java – является шаблоном, который описывает данные и поведение, связанное с экземплярами данного класса.
- После формирования класса, можно создавать объекты данного класса.



# Классы и объекты

- **Экземпляр** класса - это описание конкретного объекта в памяти.
- Каждый раз при создании **экземпляра** класса создаётся новый объект.
- Объект содержит собственную копию экземпляров переменных, определённых в классе.

# Классы и объекты

```
class Name {  
    ...  
}
```

- Все методы должны быть определены внутри класса.
- Все классы в Java наследуются от класса **Object**, поэтому для любого объекта вы можете использовать методы этого класса.

# Определение методов

- **Метод** определяется, как действительная реализация некоторой операции с объектом.

модификатор\_доступа тип название\_метода (список\_параметров)

{

// тело метода

}

- Методы возвращают значения в вызывающую программу с помощью оператора **return**.
- Описание метода может начинаться с модификаторов *public*, *protected*, *private*, *abstract*, *static*, *final*, *synchronized*, *native*.

# Примеры

- Создание объектов:

```
Button b = new Button("Enter");
```

```
MyClass obj = new MyClass();
```

- Классы инкапсулируют переменные и методы – члены класса.
- Переменные объявляются следующим образом:

спецификаторы тип имяПеременной;

- Объявление методов:

спецификаторы тип имяМетода (список\_параметров){

// тело метода

return value; // в случае возврата значения

}

MyClass
int a float b
add() insert()



# Примеры

- Вызов методов осуществляется из объекта или класса (для статических методов) в следующем виде:

имяОбъекта.имяМетода();

имяКласса.имяМетода();

# Конструктор класса

- **Конструктор** класса (class constructor) – специальный метод, который не имеет возвращаемого значения и имеет то же самое название, что и класс.
- Конструктор выполняется автоматически при создании экземпляра класса, после распределения памяти и обнуления полей, но до начала использования создаваемого объекта.
- Конструктор не возвращает никакого значения. Поэтому в его описании не пишется даже слово **void** , но можно задать модификаторы доступа.

# Конструктор класса

Определение класса на языке **Java** с помощью оператора class:

```
class MyClass {  
    // СВОЙСТВО  
    String name = "Example";  
  
    // конструктор  
    public MyClass (String name) {  
        this.name = name;  
    }  
  
    // метод  
    public String getName() {  
        return name;  
    }  
}
```

# Объектная инициализация

- В начале создания объекта JVM распределяет достаточное место в динамической области памяти, чтобы разместить переменные образца объекта.
- Когда память распределена, в ней могут содержаться различные типы данных.
- Значения присвоенные по умолчанию:

boolean – false

byte – (byte)0

int – 0

char - \u0000

double – 0.0 D

float – 0.0 f

object reference - null

# Классы

Создание экземпляра класса:

```
MyClass my = new MyClass("Example 2");
```

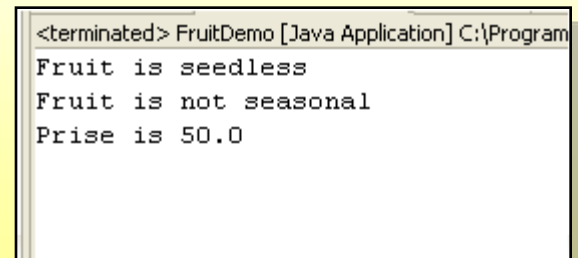
Уничтожение экземпляра класса:

- происходит с помощью "сборщика мусора" автоматически.

```
class Fruit {
    boolean seedless;
    boolean seasonal;
    float prise;

    void setProperties (boolean seed, boolean season, float p){
        seedless=seed;
        seasonal= season;
        prise=p;
    }
    void printProperties ( ){
        if (seedless) System.out.println ("Fruit is seedless");
        else System.out.println ("Fruit is seeded");
        if (seasonal) System.out.println ("Fruit is seasonal");
        else System.out.println ("Fruit is not seasonal");
        System.out.println ("Prise is "+prise);
    }
}

class FruitDemo {
    public static void main(String [] args) {
        Fruit f= new Fruit();
        f.setProperties(true, false, 50.0f);
        f.printProperties();
    }
}
```



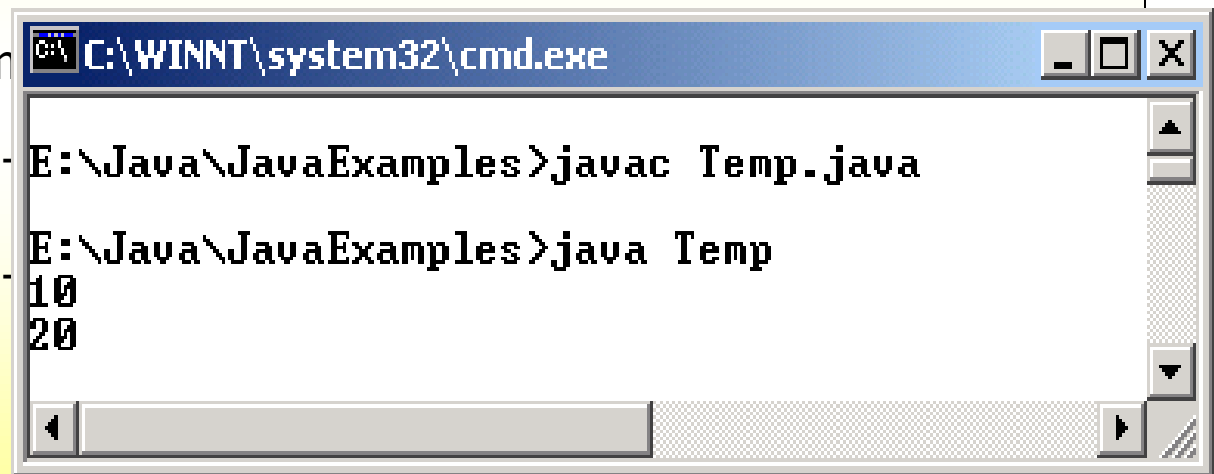
```
<terminated> FruitDemo [Java Application] C:\Program
Fruit is seedless
Fruit is not seasonal
Prise is 50.0
```

# Определение методов

- Имя метода, число и типы параметров образуют **сигнатуру метода**.
- Компилятор различает методы не по их именам, а по сигнатурам.
- Это позволяет записывать разные методы с одинаковыми именами, различающиеся числом и/или типами параметров.

# Пример использования метода

```
class Temp
{
    static int num = 10;
    public static void show()
    {
        System.out.println(num);
    }
    public static void main()
    {
        Temp t1 = new Temp();
        t1.show();
        Temp t2 = new Temp();
        t2.num = 20;
        t2.show();
    }
}
```



The screenshot shows a Windows command prompt window titled "C:\WINNT\system32\cmd.exe". The window displays the following commands and output:

```
E:\Java\JavaExamples>javac Temp.java
E:\Java\JavaExamples>java Temp
10
20
```

The output shows the number 10 on the first line and the number 20 on the second line, corresponding to the two calls to the `show()` method in the `main` method of the `Temp` class.



# Инкапсуляция

- **Инкапсуляция** является одной из основных характеристик объектно-ориентированного языка программирования.
- Суть инкапсуляции состоит в том, что в рамках класса объединяются свойства и методы и получившийся объект рассматривается как единое целое.

# Инкапсуляция

- Существует возможность назначить права доступа к объектам класса и/или составным частям объектов с использованием модификаторов.
- **Модификаторы** – ключевые слова, которые дают дополнительное значение коду и классам.

# Инкапсуляция

- Иногда объекту-получателю бывает необходимо знать ссылку на самого себя.
- Например, объект-получатель может захотеть внести себя в какой-нибудь список объектов.
- В каждом методе может использоваться ***this*** — ссылка на текущий объект.
- Ссылка ***this*** часто используется в качестве параметра для тех методов, которым нужна ссылка на объект.

# Ссылка this

- Ключевое слово **this** связано с объектом.
- Оно используется, чтобы указать текущий объект.

```
int x;
```

```
PassTest (int x) {  
    this.x=x;  
}
```

# Наследование

- **Наследование** - приобретение свойства другого объекта или класса.
- Наследование в Java осуществлено отношениями подкласса или класса.
- Суть наследования состоит в создании производного класса, который имеет возможности базового класса плюс свои собственные возможности.

# Наследование

- **Суперкласс** – это класс, от которого будет происходить наследование.
- **Подкласс** – это класс, который наследовал качества суперкласса.
- Когда наследование осуществлено, подкласс получает свойства суперкласса плюс его собственные свойства.

# Наследование

- Говорят, что подкласс **расширяет** суперкласс.
- Чтобы выполнить наследование класса, необходимо использовать ключевое слово **extends** в определении подкласса.

```
class MyApplet extends Applet {  
    ...  
}
```

- Любой подкласс может расширять только один суперкласс.

# Наследование

- Объявление **класса** с модификатором **final** означает, что его нельзя расширять или породить от него подкласс.
- По причине предотвращения наследования, все методы в классе **final** неявно тоже **final**.
- Если **метод** объявлен с модификатором **final**, значит, он объявлен окончательно и не может быть переопределен.
- Ключевое слово **final** по отношению к **переменной** означает, что значение данной переменной никогда не меняется, т.е. является константой.



# Наследование

- При использовании переопределенных методов в производных классах может возникнуть необходимость явно вызвать функцию базового класса.

`super.имяМетода(параметры);`

- Ключевое слово **super** предназначено для вызова конструктора суперкласса.
- Ключевое слово **super** может быть также использовано для обращения к методам или экземплярам переменных суперкласса.

# Полиморфизм

- **Полиморфизм** – это свойство использования одного имени для задания общих для класса действий.
- Выполнение каждого конкретного действия будет определяться типом данных.

# Перегруженные методы

- **Перегруженные методы** – это методы, которые находятся в том же самом классе и имеют то же самое имя, но различные списки параметров.
- Как и другие методы мы можем перегружать конструктор.
- Перегрузка конструкторов обеспечивает способ создания объектов с или без начальных параметров, в зависимости от необходимости.

# Типы методов

- **Перегруженные методы** – это методы, которые находятся в том же самом классе и имеют то же самое имя, но различные списки параметров.

```
public void add (int a, int b){  
    int c=a+b;  
    System.out.println(c);  
}  
  
public void add (float a, float b){  
    float c=a+b;  
    System.out.println(c);  
}
```

- **Переопределенные методы** – это методы, которые находятся в суперклассе так же как и в подклассе.

# Замещение и перегрузка методов

- Характеристики переопределенных методов:
  - ❖ Определяются как в суперклассе, так и в подклассе.
  - ❖ Переопределяются в подклассе.
- Характеристики перегружаемых методов:
  - ❖ Определены в одном и том же классе.
  - ❖ Имеют одно и то же имя.
  - ❖ Имеют различные списки параметров.

# Перегрузка конструктора

- Перегрузка конструктора обеспечивает способ создания объектов с или без начальных параметров, в зависимости от необходимости.

```
public class PassTest {  
    PassTest () {  
        ....  
    }  
    PassTest (int x) {  
        ....  
    }  
    ...  
    PassTest test1= new Passtest();  
    PassTest test2=new Passtest (10);  
}
```

# Пакеты

- Все классы Java распределяются по **пакетам**.
- **Пакет**- это совокупность классов, интерфейсов и/или других пакетов.
- Кроме классов пакеты могут включать в себя интерфейсы и вложенные **подпакеты** (subpackages).
- Каждый пакет образует одно пространство имен.
- Это означает, что все имена классов, интерфейсов и подпакетов в пакете должны быть уникальны.
- Имена в разных пакетах могут совпадать, но это будут разные программные единицы.

# Пакеты

- Чтобы создать пакет надо в первой строке Java-файла с исходным кодом записать строку:

`package Имя;`

- Имя подпакета уточняется именем пакета.  
Создание подпакета:

`package ИмяПакета.ИмяПодпакета;`

- Чтобы правильно использовать члены пакета, нужно явно включить пакет в программу Java с помощью команды *import*.

`import package_name.*;`



# Вопросы

- Что такое Java – класс?
- Какие члены классы Вы знаете? Дайте им определения.
- Что такое объект, экземпляр класса? Приведите примеры.
- Каким образом осуществляется вызов методов?
- Что такое конструктор класса и для чего он необходим?
- С помощью какого оператора методы возвращают значения в вызывающую программу?
- Что такое Java-пакеты? Приведите преимущества их использования.