



ОНЛАЙН-ОБРАЗОВАНИЕ

## 03 – Java Basics (Часть 2)

**Дмитрий Коган**



## Как меня слышно и видно?



Если нет – напишите, если слышите – смайлик в чат.



## Цели :

- **Продолжим изучать структуру Java-класса**
- **Разберёмся с пакетами и импортами**
- **Набьём руку на примерах**





**Начинаем?**

# Темы экзамена

## ☐ **Java Basics**

- ☐ Working with Java Data Types
- ☐ Using Operators and Decision Constructs
- ☐ Creating and Using Arrays
- ☐ Using Loop Constructs
- ☐ Working with Methods and Encapsulation
- ☐ Working with Inheritance
- ☐ Handling Exceptions
- ☐ Working with Selected classes from the Java API

# Подтемы экзамена

## Java Basics

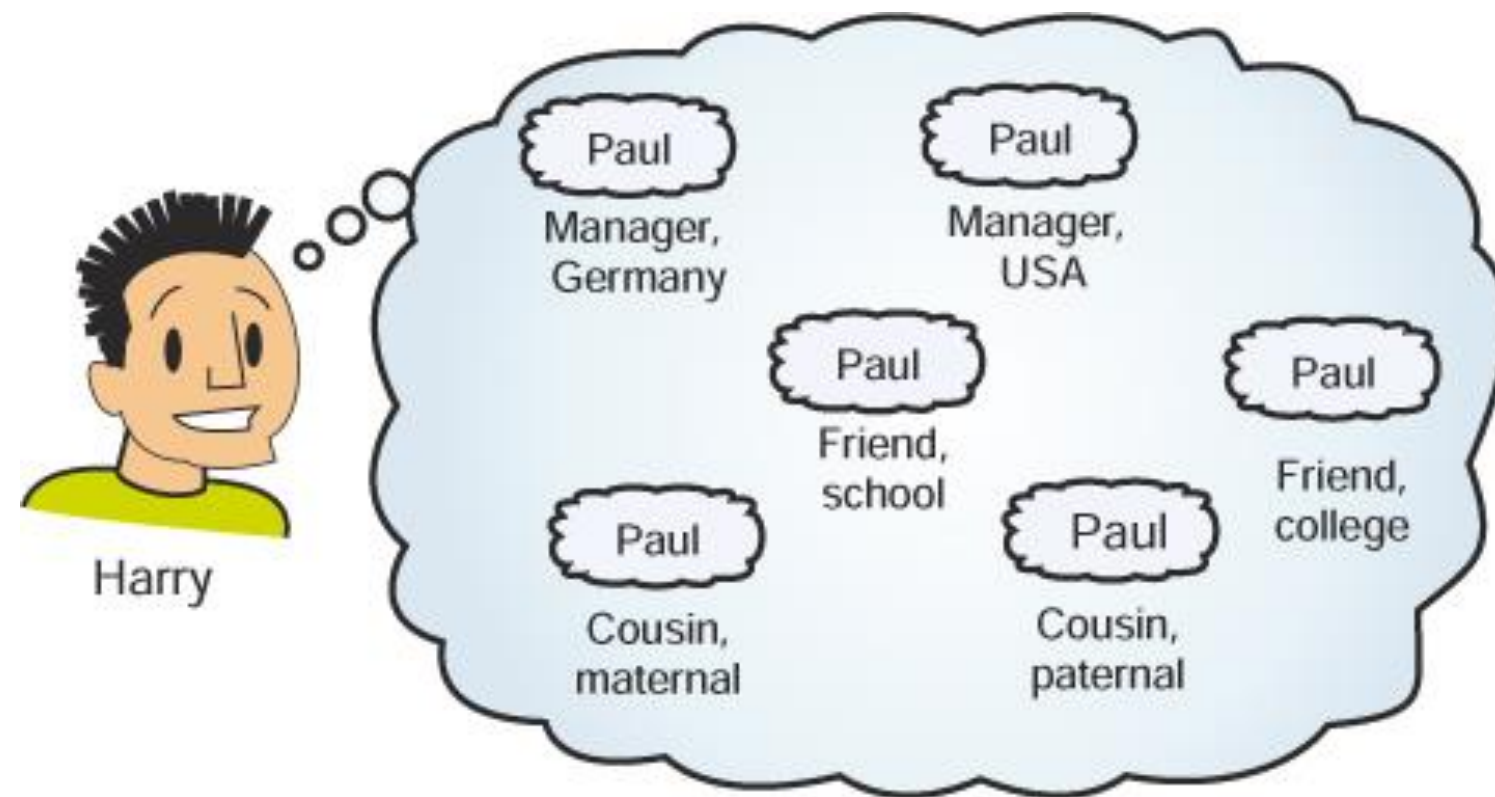
- Define the scope of variables
- **Define the structure of a Java class**
- Create executable Java applications with a main method; **run a Java program from the command line;** produce console output
- **Import other Java packages to make them accessible in your code**
- Compare and contrast the features and components of Java such as: platform independence, object orientation, encapsulation, etc.



# Пакеты Java



# Пакеты



6 знакомых Гарри

# Пакеты

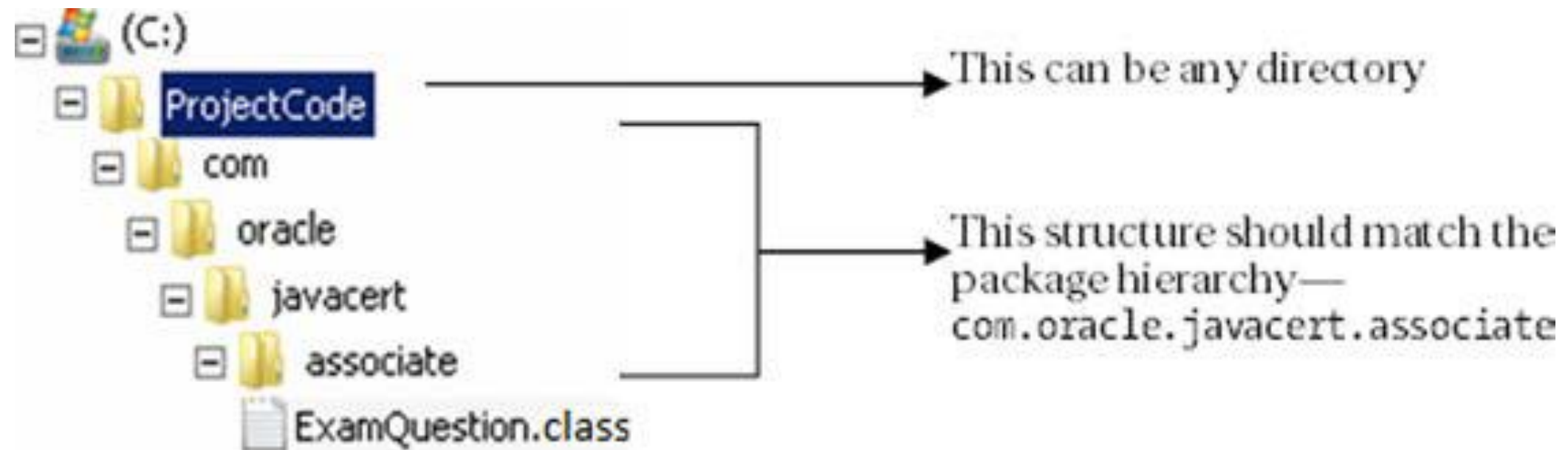
```
package com.oracle.javacert.associate;  
class ExamQuestion {  
    // variables and methods  
}
```

com.oracle.javacert.associate

ExamQuestion

Package or subpackage name	Its meaning
com	Commercial. A couple of the commonly used three-letter package abbreviations are <ul style="list-style-type: none"><li>▪ gov—for government bodies</li><li>▪ edu—for educational institutions</li></ul>
oracle	Name of the organization
javacert	Further categorization of the project at Oracle
associate	Further subcategorization of Java certification

# Пакеты

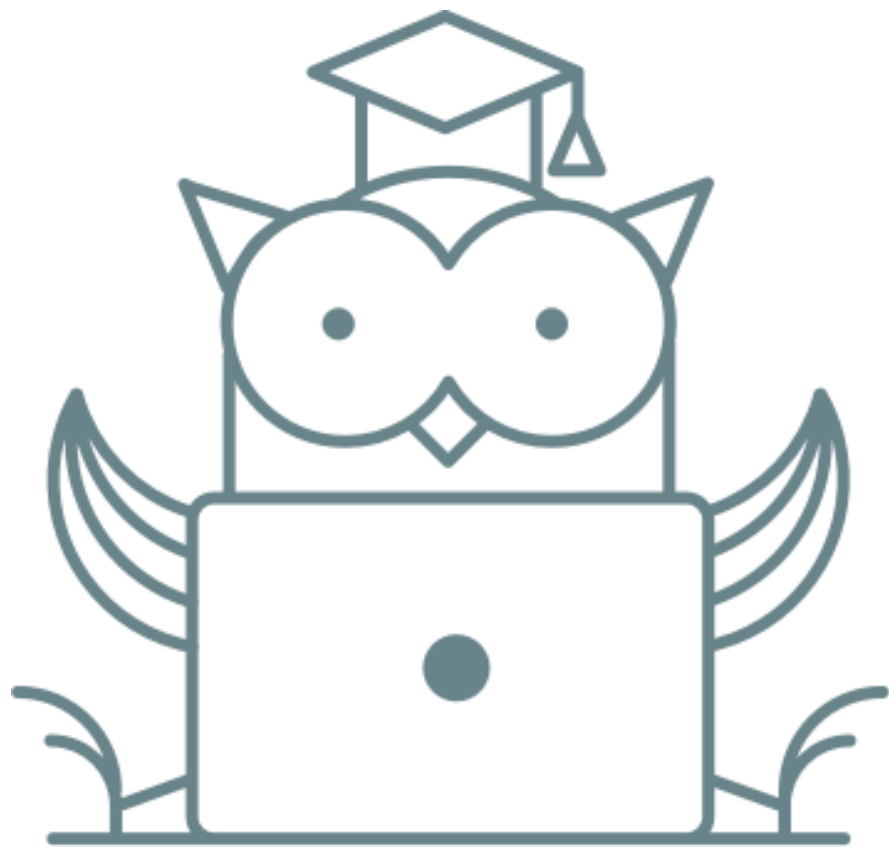


# Пакеты

- ✓ Функционально родственные классы и интерфейсы принято хранить в пакетах. Этот подход обеспечивает также защиту доступа и управление пространством имен.
- ✓ Если декларация `package` присутствует, она должна быть первой незакомментированной строкой в исходном `.java`-файле. Любое иное размещение, например внутри тела класса или после него, ведет к ошибке компиляции.
- ✓ В исходном файле может содержаться только одна декларация `package`.
- ✓ По умолчанию, классы и интерфейсы в разных пакетах не видны друг для друга. И напротив, все классы и интерфейсы в пределах одного пакета видны друг другу.
- ✓ Имена пакета и подпакетов соединяются посредством оператора «точка».

# Пакеты по умолчанию

- ✓ Если файл с исходным кодом не содержит package-декларацию, класс / интерфейс считается членом т.н. дефолтного пакета.
- ✓ Члены дефолтного пакета доступны лишь тем классам или интерфейсам, которые находятся в этой же файловой папке.
- ✓ Дата-типы из дефолтного пакета не доступны дата-типам, привязанным к какому-то конкретному пакету, пусть даже все они находятся в одной и той же файловой папке.



**Вопросы?**



## Импорт классов

# Импорта нет

```
public class ImportExample {  
    public static void main(String[] args) {  
        Random r = new Random();    // DOES NOT COMPILE  
        System.out.println(r.nextInt(10));  
    }  
}
```

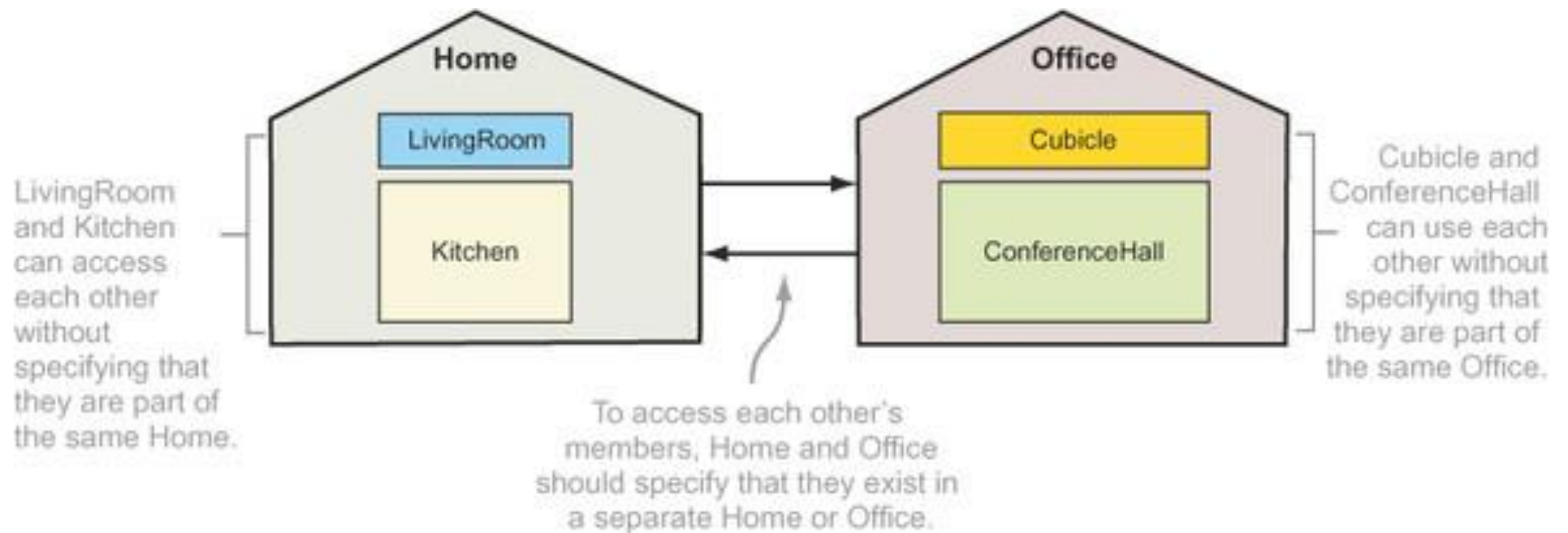
Random cannot be resolved to a type



# Импорт есть

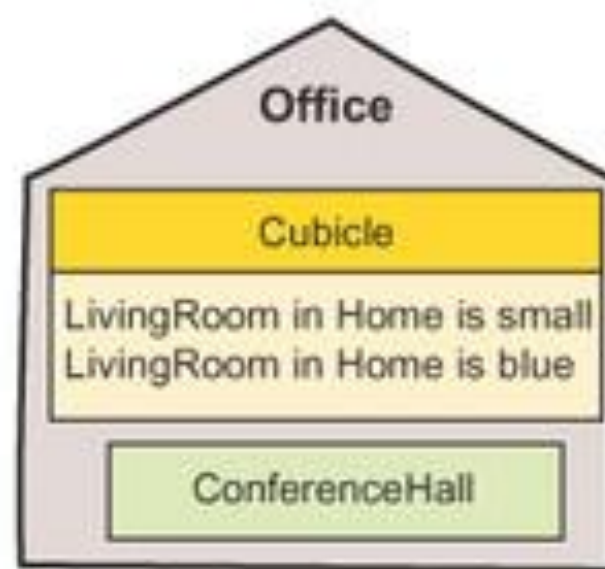
```
import java.util.Random; // import tells us where to find Random
public class ImportExample {
    public static void main(String[] args) {
        Random r = new Random();
        System.out.println(r.nextInt(10)); // print a number between 0 and 9
    }
}
```

# Имена классов

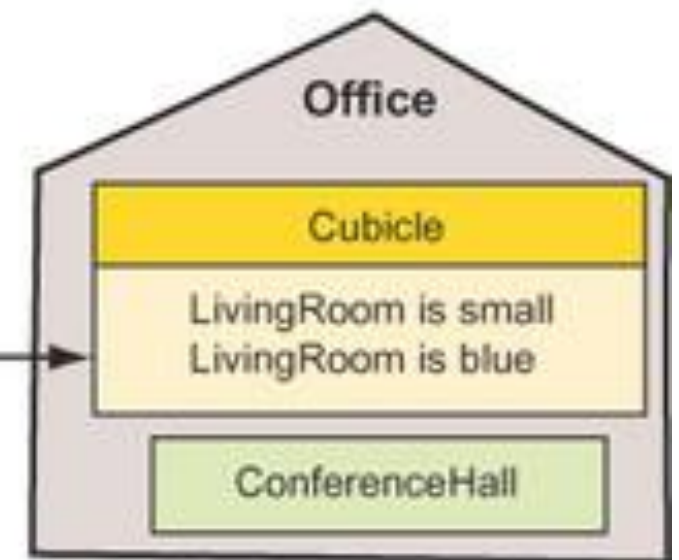
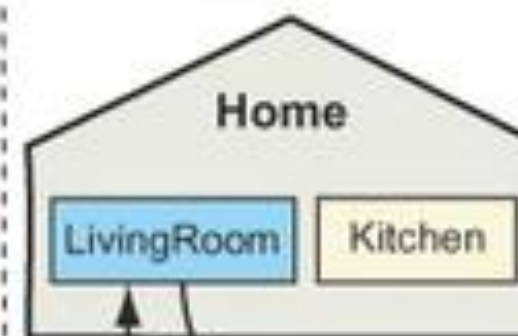


# Имена классов

No import = use fully qualified names



Import = use simple names



Import LivingRoom  
in Cubicle.

LivingRoom is still in  
Home. It is not  
embedded in  
Cubicle.

# Имена классов

```
package office;  
class Cubicle {  
    home.LivingRoom livingRoom;  
}
```

← In the absence of an import statement, use the fully qualified name to access class LivingRoom.

```
package office;  
import home.LivingRoom;  
class Cubicle {  
    LivingRoom livingRoom;  
}
```

← **import  
statement**

← No need to use the fully qualified name of class LivingRoom

# Wildcard (Звёздочка)

```
import java.util.*;    // imports java.util.Random among other things
public class ImportExample {
    public static void main(String[] args) {
        Random r = new Random();
        System.out.println(r.nextInt(10));
    }
}
```

# Избыточные импорты

```
1: import java.lang.System;
2: import java.lang.*;
3: import java.util.Random;
4: import java.util.*;
5: public class ImportExample {
6:     public static void main(String[] args) {
7:         Random r = new Random();
8:         System.out.println(r.nextInt(10));
9:     }
10: }
```

# Избыточные импорты

```
public class InputImports {  
    public void read(Files files) {  
        Paths.get("name");  
    }  
}
```

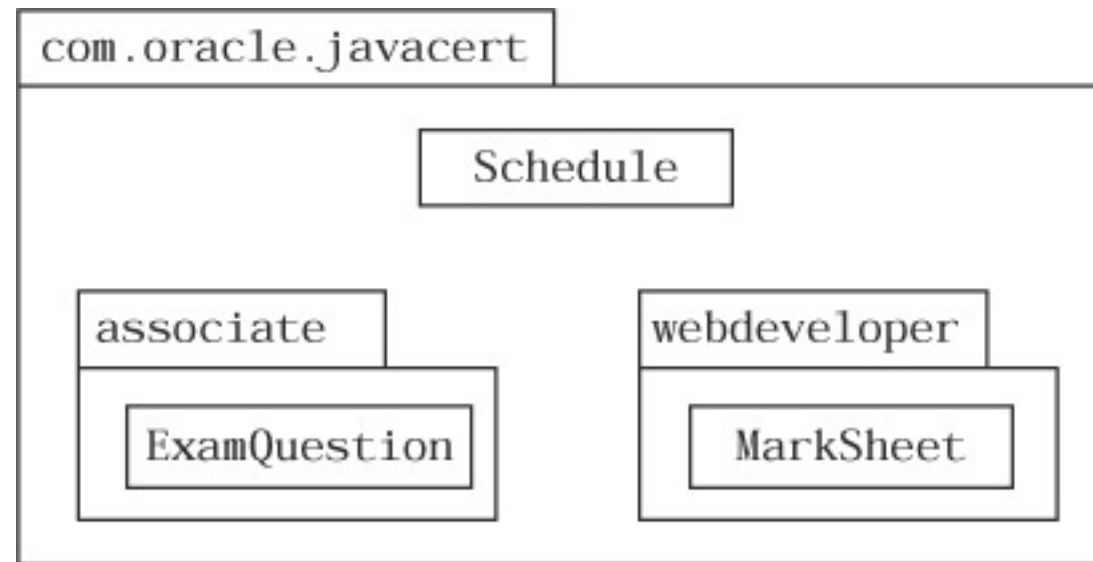
```
import java.nio.file.*;
```

The other answer is to import both classes explicitly:

```
import java.nio.file.Files;  
import java.nio.file.Paths;
```

```
import java.nio.*; // NO GOOD - a wildcard only matches  
                    //class names, not "file.*Files"  
import java.nio.*.*; // NO GOOD - you can only have one wildcard  
                    //and it must be at the end  
import java.nio.files.Paths.*; // NO GOOD - you cannot import methods  
                    //only class names
```

# Импорт подкаталогов



```
import com.oracle.javacert.*;
```

← Imports the class  
Schedule only

```
import com.oracle.javacert.associate.*;  
import com.oracle.javacert.webdeveloper.*;
```

← Imports class  
ExamQuestion only

← Imports class  
MarkSheet only



# Пакеты по умолчанию

```
class Person {  
    // code  
}  
class Office {  
    Person p;  
}
```

**Not defined in an  
explicit package**

← **Class Person accessible  
in class Office**

**Классы должны находиться в одном каталоге!**

# Конфликты имён

```
public class Conflicts {  
    Date date;  
    // some more code  
}
```

```
import java.util.*;  
import java.sql.*;
```

# Конфликты имён

```
public class Conflicts {  
    Date date;  
    // some more code  
}
```

```
import java.util.*;  
import java.sql.*; // DOES NOT COMPILE
```

The type Date is ambiguous

# Конфликты имён

```
public class Conflicts {  
    Date date;  
    // some more code  
}
```

```
import java.util.Date;  
import java.sql.*;
```

```
import java.util.Date;  
import java.sql.Date;
```

The `import java.sql.Date` collides with another import statement

# Использование классов с одинаковым именем

```
import java.util.Date;

public class Conflicts {
    Date date;
    java.sql.Date sqlDate;
}
```

Or you could have neither with an import and always use the fully qualified class name:

```
public class Conflicts {
    java.util.Date date;
    java.sql.Date sqlDate;
}
```

# Пакет `java.lang`

- ✓ В отличие от всех прочих пакетов, пакет

`java.lang`

автоматически импортируется компилятором.

- ✓ Пакет содержит фундаментальные дата-типы, системные методы и наиболее популярные классы, в частности

`String`, `StringBuilder`, `Math` и т.д.,  
подпроцессы (`threads`) и  
исключения (`exceptions`).

# Ловушка на экзамене

Если дата-тип не `public`, его импорт невозможен:

```
package pack1;
class A {}

package pack2;
import pack1.*;
// import pack1.A;                // INVALID: "A is not public in pack1; cannot
                                   // be accessed from outside package"

// class B extends A {}           // INVALID: та же история
// class C extends pack1.A {}     // INVALID: -"-
```

# Упражнение (начало)

## Janitor.java:

```
package office;
public class Janitor {
    // some valid code
}
```

## BroomCloset.java:

```
package office;
public class BroomCloset {
    // lot and lot of valid code
}
```

## Broom.java:

```
package office.broomcloset;
public class Broom {
    // even bigger amount of valid and astonishingly crafty code with all
    // necessary constructors and even a mahogany handle for the broom
}
```



# Упражнение (окончание)

## AngryBoss.java:

```
package empirestatebuilding;

// INSERT your code here

public class AngryBoss {
    public Broom fetchBroom(Janitor j, BroomCloset bc) {
        // valid code
        return new Broom();
    }
}
```

**Which LOC(s), when inserted in AngryBoss.java, shall enable the code to compile?**

- A. `import office.*.*;`
- B. `import office.broomcloset.*;`
- C. `import office;`  
`import office.broomcloset;`
- D. `import office.BroomCloset;`  
`import office.broomcloset.Broom;`
- E. `import office.*;`  
`import office.broomcloset.*;`

# Упражнение

## A.java:

```
import java.lang.*;
import java.lang.String;
public class A {
    public void runMe(String str) {}
    String a = str;
}
```

## B.java:

```
public class B {
    public void answer() {
        private String p = "protected";
        System.out.println("This StackOverflow question is " + p);
    }
}
```

## C.java:

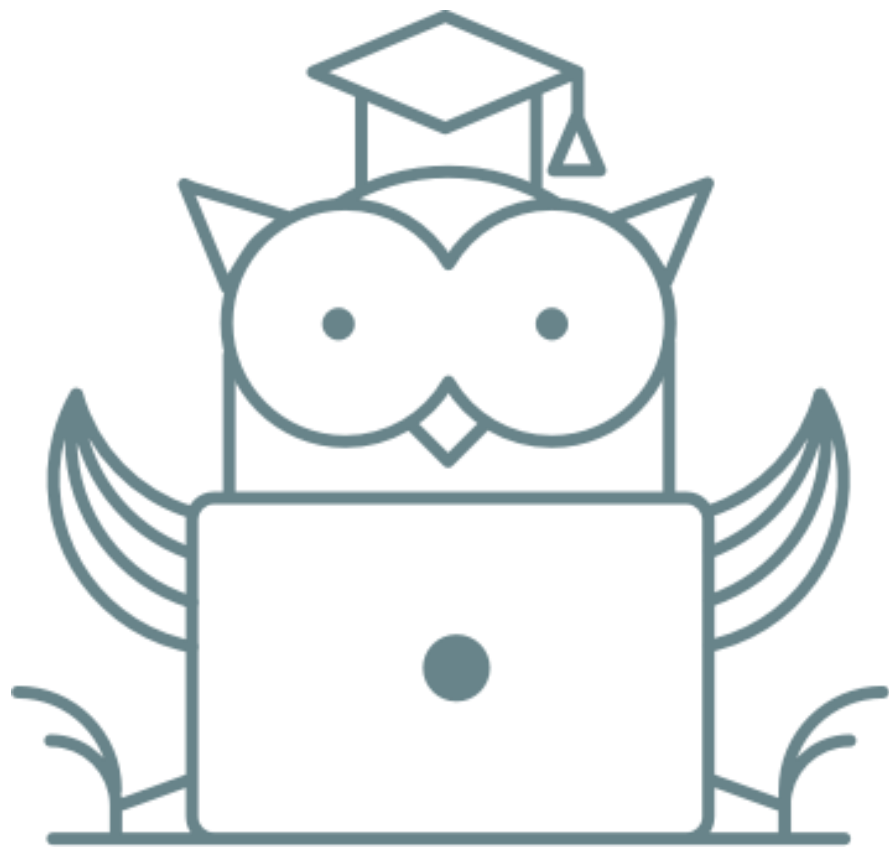
```
import java.io.FileNotFoundException;
package mypack;
interface FileReader {
    String readFromFile(String fname) throws FileNotFoundException;
}
```

## Which statement is true?

- A. Only A.java compiles successfully.
- B. Only B.java compiles successfully.
- C. Only C.java fails compilation.
- D. All three files fail compilation.



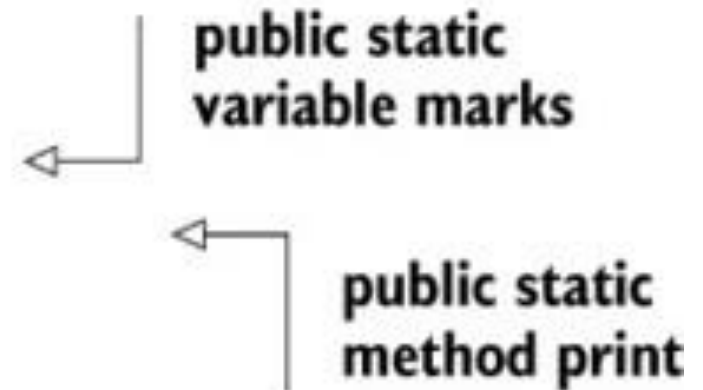
**Вопросы?**



# Статический импорт

# Статический импорт

```
package certification;  
public class ExamQuestion {  
    static public int marks;  
    public static void print() {  
        System.out.println(100);  
    }  
}
```



**public static  
variable marks**

**public static  
method print**

The diagram consists of two L-shaped arrows. The first arrow originates from the text 'public static variable marks' and points to the 'static public int marks;' line in the code block. The second arrow originates from the text 'public static method print' and points to the 'public static void print()' line in the code block.

# Статический импорт

```
package university;
import static certification.ExamQuestion.marks;
class AnnualExam {
    AnnualExam() {
        marks = 20;
    }
}
```

← Access variable marks  
without prefixing it  
with its class name

← Correct statement  
is import static, not  
static import

```
package university;
import static certification.ExamQuestion.*;
class AnnualExam {
    AnnualExam() {
        marks = 20;
        print();
    }
}
```

Accesses variable marks and method print  
without prefixing them with their class names

← Imports all static  
members of class  
ExamQuestion

# Статический импорт

```
public class TestStatic {  
    public static void main(String[] args) {  
        System.out.println(Integer.MAX_VALUE);  
        System.out.println(Integer.toHexString(42));  
    }  
}
```

```
import static java.lang.System.out;           // 1  
import static java.lang.Integer.*;           // 2  
public class TestStaticImport {  
    public static void main(String[] args) {  
        out.println(MAX_VALUE);               // 3  
        out.println(toHexString(42));         // 4  
    }  
}
```

# Статический импорт

```
import static java.lang.Math.PI;           // (1) Static field
import static java.lang.Math.sqrt;         // (2) Static method
// Only specified static members are imported.

public class Calculate3 {
    public static void main(String[] args) {
        double x = 3.0, y = 4.0;
        double squareroot = sqrt(y);        // Simple name of static method
        double hypotenuse = Math.hypot(x, y); // (3) Requires type name
        double area = PI * y * y;           // Simple name of static field
        System.out.printf("Square root: %.2f, hypotenuse: %.2f, area: %.2f\n",
                           squareroot, hypotenuse, area);
    }
}
```

Square root: 2.00, hypotenuse: 5.00, area: 50.27



# Затенение

```
import static java.lang.System.out;          // (1) Static import

public class ShadowImport {

    public static void main(String[] args) {
        out.println("Calling println() in java.lang.System.out");
        ShadowImport sbi = new ShadowImport();
        writeInfo(sbi);
    }

    // Parameter shadows java.lang.System.out:
    public static void writeInfo(ShadowImport out) {
        out.println("Calling println() in the parameter out");
        System.out.println("Calling println() in java.lang.System.out"); //
Qualify
    }

    public void println(String msg) {
        out.println(msg + " of type ShadowImport");
    }
}
```

```
Calling println() in java.lang.System.out
Calling println() in the parameter out of type ShadowImport
Calling println() in java.lang.System.out
```

# Конфликт имён

```
import static java.lang.Integer.MAX_VALUE;
import static java.lang.Double.MAX_VALUE;

public class StaticFieldConflict {
    public static void main(String[] args) {
        System.out.println(MAX_VALUE);           // (1) Ambiguous! Compile-time
error!
        System.out.println(Integer.MAX_VALUE);   // OK
        System.out.println(Double.MAX_VALUE);     // OK
    }
}
```

# Импорт методов с одинаковой сигнатурой

```
package mypkg;
```

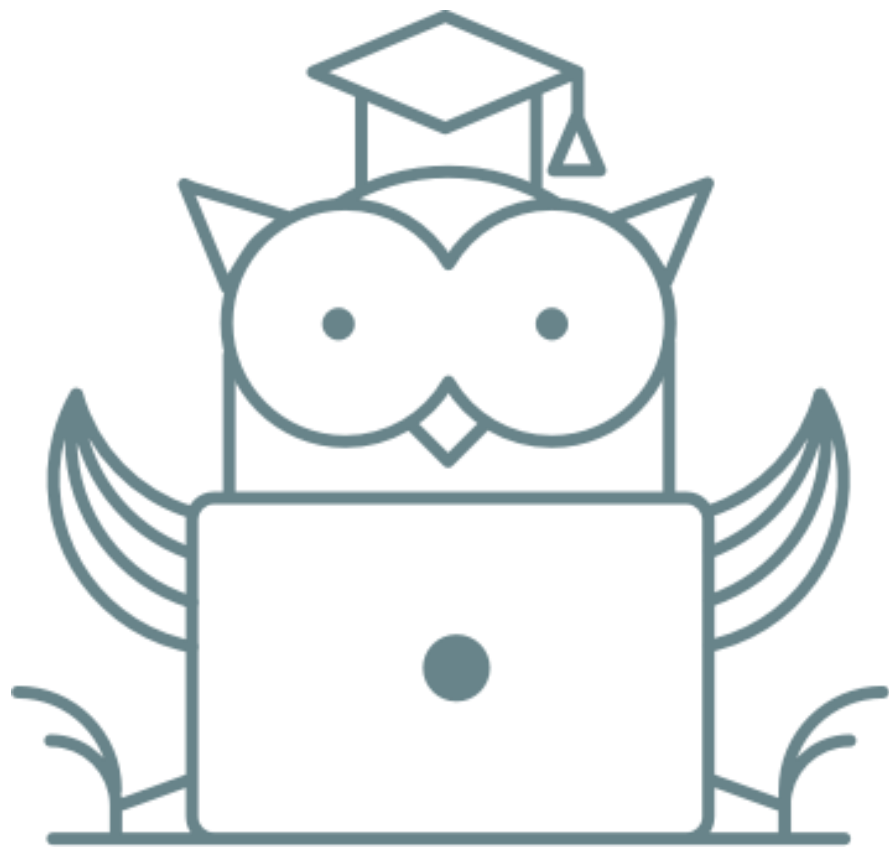
```
public class Auxiliary {  
    public static int binarySearch(int[] a, int key) { // Same in  
java.util.Arrays  
    // Implementation is omitted.  
    return -1;  
}  
}
```

---

```
// File: MultipleStaticImport.java (in unnamed package)  
import static java.util.Collections.binarySearch; // 2 overloaded methods  
import static java.util.Arrays.binarySearch; // + 18 overloaded methods  
import static mypkg.Auxiliary.binarySearch; // (1) Causes signature conflict
```

```
public class MultipleStaticImport {  
    public static void main(String[] args) {  
        int index = binarySearch(new int[] {10, 50, 100}, 50); // (2) Ambiguous!  
        System.out.println(index);  
    }  
}
```

```
//public static int binarySearch(int[] a, int key) { // (3)  
//    return -1;  
//}  
}
```



**Вопросы?**



## Запуск файлов из пакетов

# Два класса

C:\temp\packagea\ClassA.java

```
package packagea;  
public class ClassA {  
}
```

C:\temp\packageb\ClassB.java

```
package packageb;  
import packagea.ClassA;  
public class ClassB {  
    public static void main(String[] args) {  
        ClassA a;  
        System.out.println("Got it");  
    }  
}
```

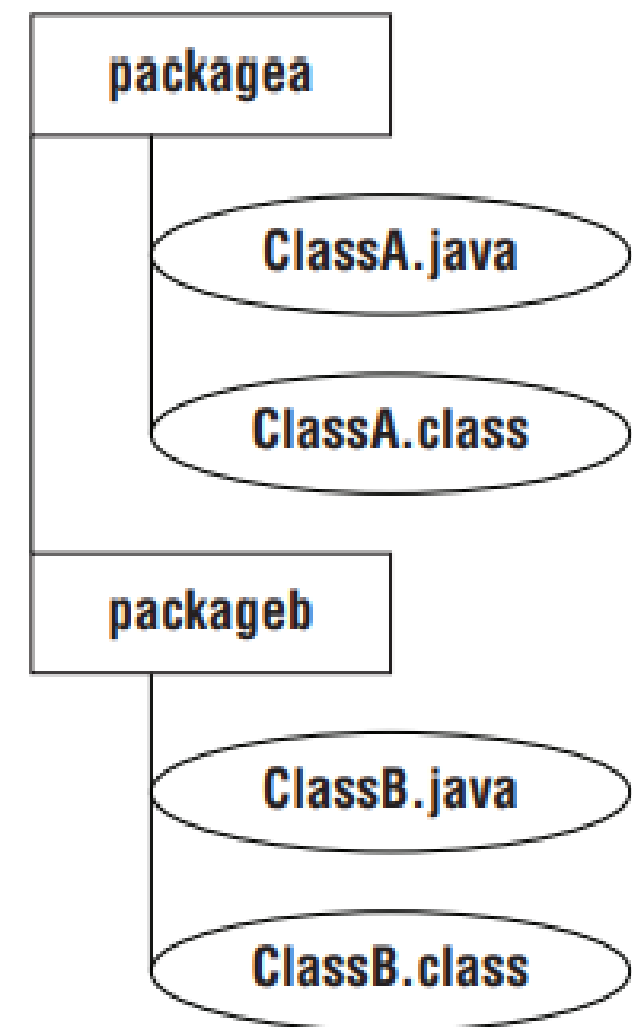
Go to directory.

cd C:\temp

# Запускаем

```
javac packagea/ClassA.java packageb/ClassB.java  
javac packagea/*.java packageb/*.java  
java packageb.ClassB
```

If it works, you'll see Got it printed.



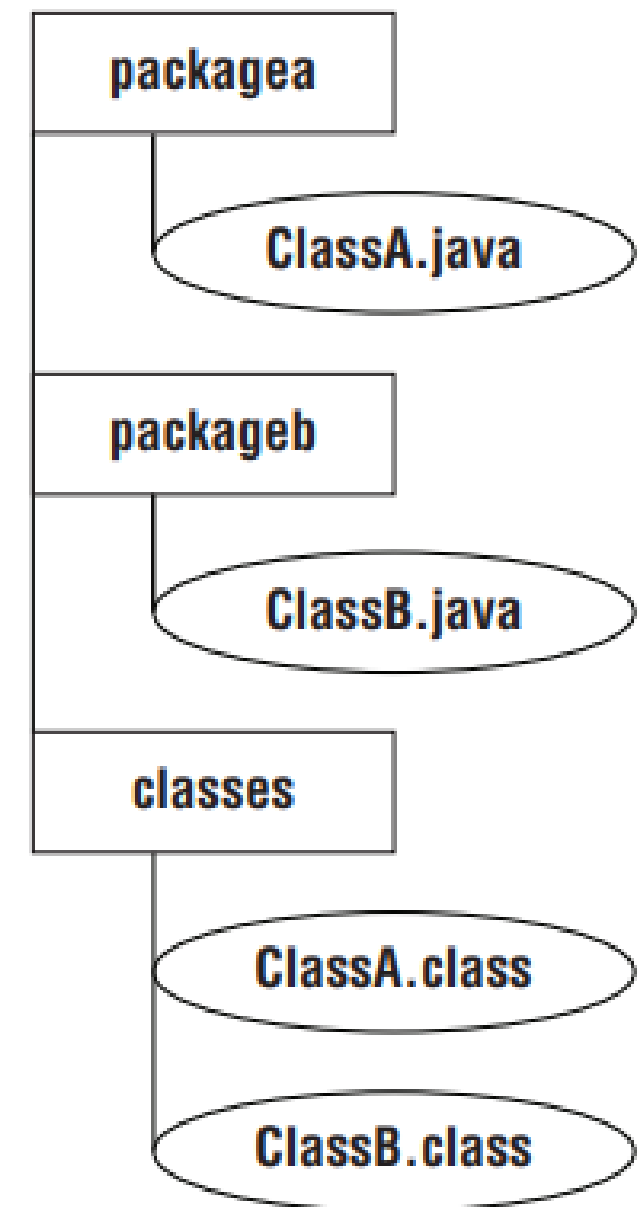
# Другая папка

```
javac -d classes packagea/ClassA.java packageb/ClassB.java
```

```
java -cp classes packageb.ClassB
```

```
java -classpath classes packageb.ClassB
```

```
java --class-path classes packageb.ClassB
```





# JAR файл

```
java -cp ".;C:\temp\someOtherLocation;c:\temp\myJar.jar" myPackage.MyClass
```

```
java -cp "C:\temp\directoryWithJars\*" myPackage.MyClass
```

# Создание JAR файла

```
jar -cvf myNewFile.jar .
```

```
jar --create --verbose --file myNewFile.jar .
```

Alternatively, you can specify a directory instead of using the current directory.

```
jar -cvf myNewFile.jar -C dir .
```

Option	Description
-c --create	Creates a new JAR file
-v --verbose	Prints details when working with JAR files
-f <fileName> --file <fileName>	JAR filename
-C <directory>	Directory containing files to be used to create the JAR

# Вспоминаем

**Given the complete contents of the file Jupiter.java:**

```
1 public class Jupiter {  
2     public static void main (String[] args) {  
3         System.out.print("Welcome " + args[1] + "!");  
4     }  
5 }  
6 class Juno {  
7     public static void main (String[] args) {  
8         Jupiter.main(args);  
9     }  
10 }
```

**And the commands:**

```
javac Jupiter.java  
java Juno Jupiter Juno
```

**What is the result?**

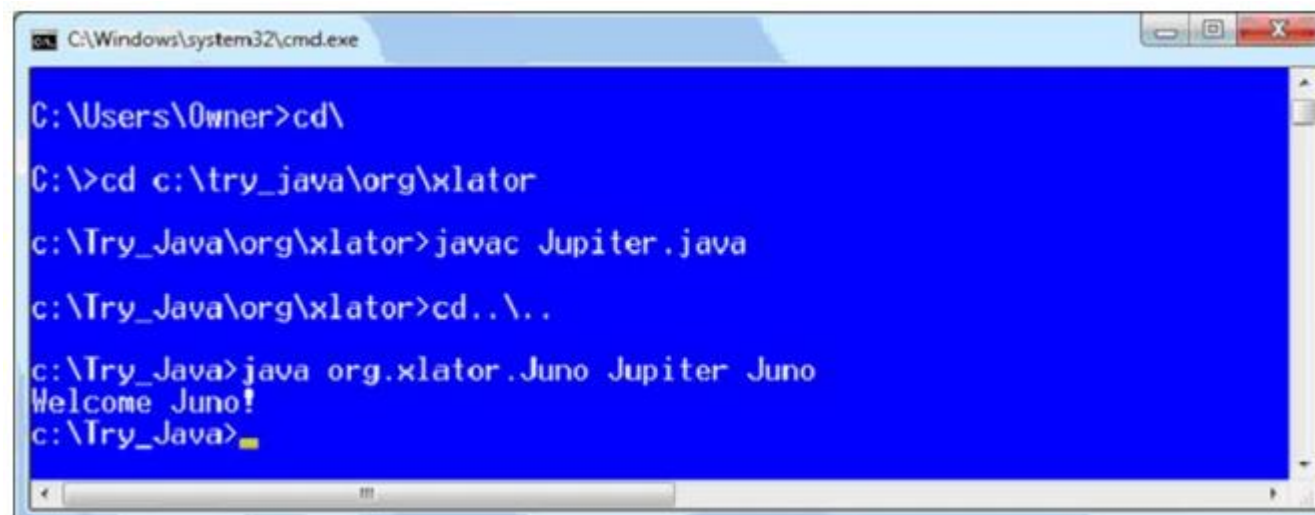
- A. Welcome Jupiter!
- B. Welcome Juno!
- C. Compilation fails because a source file can contain only one main() method
- D. An ArrayIndexOutOfBoundsException is thrown
- E. Run-time error is thrown because we run wrong class: it should've been Jupiter

# Добавляем пакеты

```
1 package org.xlator;                                // ← это единственное изменение в нашем коде
2 public class Jupiter {
3     public static void main (String[] args) {
4         System.out.print("Welcome " + args[1] + "!");
5     }
6 }
7 class Juno {
8     public static void main (String[] args) {
9         Jupiter.main(args);
10    }
11 }
```

JVM: "Could not find or load main class Juno"

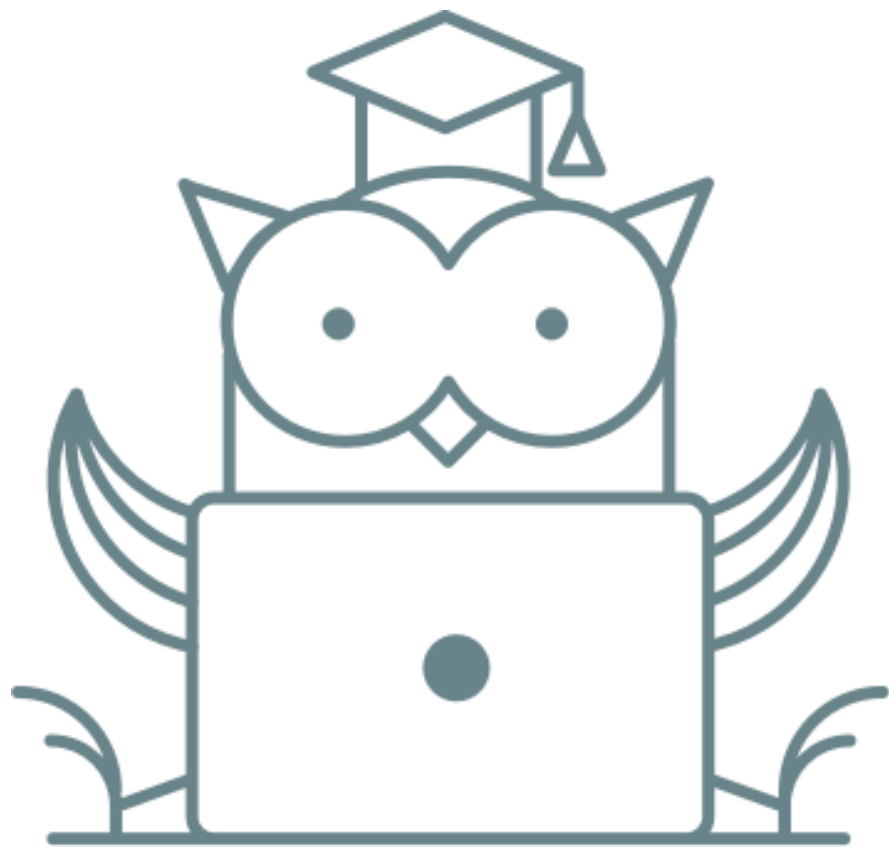
C:\Try\_Java\org\xlator



```
C:\Windows\system32\cmd.exe
C:\Users\Owner>cd\
C:\>cd c:\try_java\org\xlator
c:\Try_Java\org\xlator>javac Jupiter.java
c:\Try_Java\org\xlator>cd..\..
c:\Try_Java>java org.xlator.Juno Jupiter Juno
Welcome Juno!
c:\Try_Java>
```



**Вопросы?**



## Форматирование кода на экзамене

# Два варианта кода

```
6: public void method(ArrayList list) {  
7:   if (list.isEmpty()) { System.out.println("e");  
8: } else { System.out.println("n");  
9: } }
```

```
1: public class LineNumbers {  
2:   public void method(ArrayList list) {  
3:     if (list.isEmpty()) { System.out.println("e");  
4:   } else { System.out.println("n");  
5: } } }
```

# Будьте бдительны

- ✓ Если код выглядит чистым, а строки пронумерованы с единицы, следует быть начеку: какие-то импорты могут отсутствовать — особенно когда код упоминает:
  - списки (`List` и `ArrayList`)
  - предикативные лямбды
  - `LocalDateTime`-типы
  - класс `Arrays` (например, `Arrays.asList()`)
  - класс `Collections` (например, `Collections.sort()`)





**Вопросы?**



## **Расположение компонентов класса**

# Классовая география

- ✓ Если декларация `package` присутствует, она должна быть первой незакомментированной строкой в исходном `.java`-файле. Любое иное размещение, например внутри тела класса или после него, ведет к ошибке компиляции
- ✓ Следствие: Если указан `package`, все импорты ставятся сразу за ним
- ✓ В исходном файле может содержаться только одна декларация `package`
- ✓ Допускаются множественные декларации `import`

# Классовая география

- ✓ Поля и методы не обязательны, но в отличие от деклараций `package` и `import`, которые должны предшествовать декларации класса, поля и методы разрешается размещать внутри тела класса в любом порядке
- ✓ Комментарии могут встречаться многократно, их допускается ставить до или после `package`-декларации, вне и внутри тела класса, а также вне или внутри тела метода, конструктора, подблока или цикла

# Классовая география

- ✓ Java-класс может содержать множество членов, как то: static-переменные, поля экземпляра, методы или конструкторы, чьи объявления допускается размещать внутри класса в произвольном порядке
- ✓ Поскольку порядок следования членов жестко не задан, метод может, к примеру, обратиться к какой-нибудь объектной переменной еще до ее объявления в файле

# Пример

```
1  /* Файл: Test.java
2   * Иллюстрация принципов, регламентирующих
3   * структуру типового Джава-класса
4  */
5
6  package org.xlator;
7  import java.lang.*;
8  // package org.xlator;           // INVALID
9  // import java.util.*;         // VALID, хотя в коде реализован иной подход
10 import java.util.ArrayList;
11 import java.util.Date;
12 // import java.sql.Date;       // INVALID
13
14 interface I1{}
15 // public interface I2{}       // INVALID
16
17 class C1{ }
18 public class Test {
19     public static void main(String[] args) {
20         System.out.println(new Test().list.add("Hello")); // печатает true
21     }
22     java.util.List<String> list = new ArrayList<String>();
23 }
```

# Упражнение

**Given the contents of two Java source files:**

```
package hackathon.hacker;
public class Hacker {
    public void hack() {
        System.out.println("Done!");
    }
}
1 package hackathon;
2 public class Hackathon {
3     public static void main(String[] args) {
4         System.out.println("Ready... set... go!");
5         new Hacker().hack();
6     }
7 }
```

**What three modifications, made independently to the class Hackathon, will enable the code to compile and run?**

- A. Replace line 5 with new hackathon.hacker.Hacker().hack();
- B. Replace line 5 with new hackathon.\*.Hacker().hack();
- C. Add import hackathon.hacker.\*; before line 1
- D. Add import hackathon.hacker.\*; after line 1
- E. Add import hackathon.hacker.Hacker; after line 1



**Вопросы?**



# Домашнее задание

## Тест

- Домашнее задание

Структура Java-класса. Импорт Java-пакетов

Цель: Закрепление материала вебинара с помощью прохождения теста, аналогичного экзаменационному.

1. Пройдите, пожалуйста, тест: <https://forms.gle/64ChbhJoLboVKyi26>
2. Сообщите о прохождении в Чате с преподавателем.

Критерии оценки: Тест считается пройденным, если результат - выше 65%.

Рекомендуем сдать до: 30.04.2021

Статус: не сдано

[Чат с преподавателем](#)



**Пожалуйста, пройдите опрос**

**<https://otus.ru/polls/17809/>**



**Спасибо  
за внимание!**

**Хороших пакетов  
с импортом!**