

Лекция 10.1

Класове и Обекти Част II

Основни теми

- Употреба на *static* променливи и методи.
- Импорт и приложение на *static* членове на клас.
- Употреба на *enum* тип данни за работа с множество от константи с уникални идентификатори.
- Декларациране на *enum* константи, чрез параметри.
- Задачи

- 8.8 Принцип на композицията
- 8.9 “Събиране на боклука” и метод *finalize()*
- 8.10 *static* членове на клас
- 8.11 *static* импорт
- 8.12 *final* данни на клас
- 8.13 Изброим тип данни (*enum*)
- 8.14 Многократно използване на софтуер
- 8.15 Пример: Създаване на библиотека (*package*) за *class Time*
- 8.16 Библиотечен (*package*) достъп
- 8.17 Пример: Приложение на *class Graphics* в обекти от GUI

Задачи

Литература:

Java How to Program, Sixth Edition, глава 8

8.8 Принцип на композицията

- **Композиция**

- Даден клас е съставен (**изграден**) или има **отношение** към референции до обекти на други класове
- Нарича се още релация **ИМА** (*has-a*)

Пример (от **лекция 9.2**):

1. Един прозорец (графичен интерфейс *JFrame*) **ИМА**

- Етикет (обект от *JLabel* клас)
- Текстово поле (обект от *TextField* клас)

2. Една кола (class *Car*) **ИМА**

- Притежател (обект от *class Employee*)
- Двигател (обект от *class Engine*)

Software Engineering факти 8.9

Композицията е форма за осъществяване на многократно използване на софтуер (software reuse), при което членовете на даден клас са референции към вече дефинирани други класове.

Outline

Date.java

(1 от 3)

```
1 // Fig. 8.7: Date.java
2 // Date class declaration.
3
4 public class Date
5 {
6     private int month; // 1-12
7     private int day;    // 1-31 based on month
8     private int year;   // any year
9
10    // constructor: call checkMonth to confirm proper value for month;
11    // call checkDay to confirm proper value for day
12    public Date( int theMonth, int theDay, int theYear )
13    {
14        month = checkMonth( theMonth ); // validate month
15        year = theYear; // could validate year
16        day = checkDay( theDay ); // validate day
17
18        System.out.printf(
19            "Date object constructor for date %s\n", this );
20    } // end Date constructor
21
```



Outline

Валидира стойността
на *month*

```
22 // utility method to confirm proper month value
23 private int checkMonth( int testMonth )
24 {
25     if ( testMonth > 0 && testMonth <= 12 ) // validate month
26         return testMonth;
27     else // month is invalid
28     {
29         System.out.printf(
30             "Invalid month (%d) set to 1.", testMonth );
31         return 1; // maintain object in consistent state
32     } // end else
33 } // end method checkMonth
34
35 // utility method to confirm proper day value based on month and year
36 private int checkDay( int testDay )
37 {
38     int daysPerMonth[] =
39         { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
40
```

Date.java

(2 от 3)

Валидира
стойността на
day



Outline

```
41 // check if day in range for month
42 if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
43     return testDay;
44
45 // check for leap year
46 if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
47     ( year % 4 == 0 && year % 100 != 0 ) ) )
48     return testDay;
49
50 System.out.printf( "Invalid day (%d) set to 1.", testDay );
51 return 1; // maintain object in consistent state
52 } // end method checkDay
53
54 // return a String of the form month/day/year
55 public String toString()
56 {
57     return String.format( "%d/%d/%d", month, day, year );
58 } // end method toString
59 } // end class Date
```

Валидира дали
стойността на
day за 29
February е във
високосна
година

Date.java

(3 от 3)



Outline

Employee.java

```

1 // Fig. 8.8: Employee.java
2 // Employee class with references to other objects.
3
4 public class Employee
5 {
6     private String firstName;
7     private String lastName;
8     private Date birthDate;
9     private Date hireDate;
10
11 // constructor to initialize name, birth date and hire date
12 public Employee( String first, String last, Date dateOfBirth,
13     Date dateOfHire )
14 {
15     firstName = first;
16     lastName = last;
17     birthDate = dateOfBirth;
18     hireDate = dateOfHire;
19 } // end Employee constructor
20
21 // convert Employee to String format
22 public String toString()
23 {
24     return String.format( "%s, %s Hired: %s Birthday: %s",
25         lastName, firstName, hireDate, birthDate );
26 } // end method toString
27 } // end class Employee

```

Employee има private
референции към два Date
обекта

По- правилно е да се дефинира copy
конструктор за class Date и тези две
инициализации се заменят с

```

birthDate = new Date( dateOfBirth );
hireDate = new Date( dateOfHire );

```

Неявно извиква toString метод за
обектите hireDate birthDate



Outline

EmployeeTest.java

```
1 // Fig. 8.9: EmployeeTest.java
2 // Composition demonstration.
3
4 public class EmployeeTest
5 {
6     public static void main( String args[] )
7     {
8         Date birth = new Date( 7, 24, 1949 );
9         Date hire = new Date( 3, 12, 1988 );
10        Employee employee = new Employee( "Bob", "Blue", birth, hire );
11
12        System.out.println( employee );
13    } // end main
14 } // end class EmployeeTest
```

Създава обект от клас **Employee**

Извежда обект **Employee**

Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Blue, Bob Hired: 3/12/1988 Birthday: 7/24/1949



8.9 “Събиране на боклука” и метод *finalize()*

- Събиране на боклука
 - JVM маркира даден обект като “боклук” когато няма референции на променливи към този обект
 - JVM освобождава паметта заемана от “боклук” при достигане на определено нива за зетост на паметта
- **finalize()** метод
 - Всички класове имат **finalize** метод
 - наследен е от `class Object`
 - **finalize()** се извиква при събиране на боклука
 - **finalize()** няма аргументи и връща тип `void`

Software Engineering факти 8.10

Всеки клас, който ползва ресурси на системата като файлове върху диск, мрежови връзки и др. трябва да има метод за освобождаване на тези ресурси.

Например, много от Java API класовете имат `close()` или `dispose()` методи за тази цел.

**`class Scanner`
(java.sun.com/j2se/5.0/docs/api/java/util/Scanner.html) има `close()` метод, който трябва да се изпълни веднага щом приключи четенето от съответния файл (стандартен вход, текстов файл и пр.).**

8.10 static клас членове

- **static** данни на клас
 - Наричат се също **клас променливи**, защото са обща за всички обекти на класа (*class data members*)
 - За разлика от клас променливите, **променливите на обектите** от даден клас са уникални копия за всеки такъв обект (*instance data members*)
 - Използват се :
 - Когато е необходимо всички обекти на даден клас да ползват едно и също копие на клас данна или
 - Когато дадена данна трябва да е достъпна за ползване, дори когато няма нито един обект създаден от класа или е забранено да се създават обекти от този клас
 - Достъпни са посредством името на класа с използване на разделител точка (.)
 - Трябва да се инициализират на мястото на декларацията им или компилаторът задава стойности по подразбиране

8.10 static инициализация

```
public class SomeClass
{
    private static double rate = 1.2;
    private static int[] arr = new int[3];
    static
    {
        for (int k = 0; k<arr.length; k++)
            arr[k] = k;
    }
    // other class methods ...
}
```

Software Engineering факти 8.11

Използвайте `static` променлива, когато **всички обекти** трябва да **имат** достъп до **обща** за тях данна.

Software Engineering факти 8.11

Инициализацията на *static* данните
предхожда изпълнението на конструктора

Software Engineering факти 8.12

static членове (данни и методи) на клас съществуват и могат да се използват дори, когато няма създадени обекти от дадения клас.

Outline

Employee.java

(1 от 2)

```
1 // Fig. 8.12: Employee.java
2 // Static variable used to maintain a count of the number of
3 // Employee objects in memory.
4
5 public class Employee
6 {
7     private String firstName;
8     private String lastName;
9     private static int count = 0; // number of objects in memory
10
11     // initialize employee, add 1 to static count and
12     // output String indicating that constructor was called
13     public Employee( String first, String last )
14     {
15         firstName = first;
16         lastName = last;
17
18         count++; // increment static count of employees
19         System.out.printf( "Employee constructor: %s %s; count = %d\n",
20             firstName, lastName, count );
21     } // end Employee constructor
22
```

Деклариране на **static** променлива

Инкрементира **static** променлива



Outline

Employee.java

(2 of 2)

```
23 // subtract 1 from static count when garbage
24 // collector calls finalize to clean up object;
25 // confirm that finalize was called
26 protected void finalize() ← Декларира finalize метод
27 {
28     count--; // decrement static count of employees
29     System.out.printf( "Employee finalizer: %s %s; count = %d\n",
30         firstName, lastName, count );
31 } // end method finalize
32
33 // get first name
34 public String getFirstName()
35 {
36     return firstName;
37 } // end method getFirstName
38
39 // get last name
40 public String getLastName()
41 {
42     return lastName;
43 } // end method getLastName
44
45 // static method to get static count value
46 public static int getCount() ← Декларира static метод getCount
47 {                                     за извеждане на static
48     return count;                   променливата count
49 } // end method getCount
50 } // end class Employee
```



Outline

EmployeeTest.java

(1 от 3)

```
1 // Fig. 8.13: EmployeeTest.java
2 // Static member demonstration.
3
4 public class EmployeeTest
5 {
6     public static void main( String args[] )
7     {
8         // show that count is 0 before creating Employees
9         System.out.printf( "Employees before instantiation: %d\n",
10             Employee.getCount() );
11
12         // create two Employees; count should be 2
13         Employee e1 = new Employee( "Susan", "Baker" );
14         Employee e2 = new Employee( "Bob", "Blue" );
15
```

Create new **Employee** objects

Извиква **static** метод **getCount** посредством името на клас **Employee**



Outline

```

16 // show that count is 2 after creating two Employees
17 system.out.println( "\nEmployees after instantiation: " );
18 system.out.printf( "via e1.getCount(): %d\n", e1.getCount() );
19 system.out.printf( "via e2.getCount(): %d\n", e2.getCount() );
20 system.out.printf( "via Employee.getCount(): %d\n",
21     Employee.getCount() );

```

Извиква **static** метода **getCount** чрез обектите

Извиква **static** метод **getCount()** извън обектите

```

23 // get names of Employees
24 system.out.printf( "\nEmployee 1: %s %s\nEmployee 2: %s %s\n\n",
25     e1.getFirstName(), e1.getLastName(),
26     e2.getFirstName(), e2.getLastName() );

```

EmployeeTest.java

```

28 // in this example, there is only one reference to each Employee,
29 // so the following two statements cause the JVM to mark each
30 // Employee object for garbage collection

```

(2 от 3)

```

31 e1 = null;
32 e2 = null;

```

Премахва референциите към обектите, JVM ги маркира като “боклук”

```

34 system.gc(); // ask for garbage collection to occur now
35

```

Извиква **static** метода **gc()** от class **System** да провери за “боклук” и освободи заеманите ресурси от него



Outline

EmployeeTest.java

(3 от 3)

```

36 // show Employee count after calling garbage collector; count
37 // displayed may be 0, 1 or 2 based on whether garbage collector
38 // executes immediately and number of Employee objects collected
39 System.out.printf( "\nEmployees after System.gc(): %d\n",
40     Employee.getCount() );
41 } // end main
42 } // end class EmployeeTest

```

Извиква **static** метода
getCount() да изведе
текущия брой обекти
от клас **Employee**

Employees before instantiation: 0
Employee constructor: Susan Baker; count = 1
Employee constructor: Bob Blue; count = 2

Employees after instantiation:
via e1.getCount(): 2
via e2.getCount(): 2
via Employee.getCount(): 2

Employee 1: Susan Baker
Employee 2: Bob Blue

Employee finalizer: Bob Blue; count = 1
Employee finalizer: Susan Baker; count = 0

Employees after System.gc(): 0



Правила за добро програмиране 8.1

Извиквайте всеки `static` метод **посредством името на класа** следвано от точка (.) за акцентиране на принадлежността на `static` метода към класа , а не към обект за избягване на двусмислие при четене на кода.

8.10 `static` клас членове ...

- `String` обектите са *immutable*
 - Събиране на `String` обекти и инициализацията им на друга стойност води до създаване на нов `String` обект
- `static` метода `gc()` на `class System`
 - Извиква явно "*събирача на боклук*" – специална *нишка* на JVM
 - Изпълнява се по подразбиране, когато нивото на заемани ресурси достигне определена стойност
- `static` методите не могат да реферират директно `non-static` клас данни и методи
 - Това включва референцията `this` под каквото и да е форма

Обичайна грешка при програмиране 8.7

Грешка при компилация възниква винаги, когато *static* метод извиква **директно** (не-*static*) метод от същия клас

Аналогично, грешка при компилация възниква когато *static* метод прави опит за директен достъп до данна на *this* обекта (текущия обект).

Обичайна грешка при програмиране 8.8

Използване на `this` референцията в `static` метод е синтактична грешка.

8.11 `static import`

- `static import` декларации
 - Позволява да се реферират `static` членовете също както, ако бяха дефинирани в класа който ги импортира като `static`
 - Пример за **единична** `static import` декларация

```
import static
    packageName.ClassName.staticMemberName;
```
 - Пример за **множество** `static import` декларации

```
import static packageName.ClassName.*;
```

Импортира всички **public** статични членове на `ClassName`

Outline

```
1 // Fig. 8.14: StaticImportTest.java
2 // Using static import to import static methods of class Math.
```

```
3 import static java.lang.Math.*;
```

static импорт на всички статични членове

```
4
```

```
5 public class StaticImportTest
```

StaticImportTest.java

```
6 {
```

```
7     public static void main( String args[] )
```

```
8     {
```

```
9         System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt( 900.0 ) );
```

```
10        System.out.printf( "ceil( -9.8 ) = %.1f\n", ceil( -9.8 ) );
```

```
11        System.out.printf( "log( E ) = %.1f\n", log( E ) );
```

```
12        System.out.printf( "cos( 0.0 ) = %.1f\n", cos( 0.0 ) );
```

```
13    } // end main
```

```
14 } // end class StaticImportTest
```

Ползва **Math**'s **static** методи без рефериране на клас **Math**.

```
sqrt( 900.0 ) = 30.0
ceil( -9.8 ) = -9.0
log( E ) = 1.0
cos( 0.0 ) = 1.0
```



Обичайна грешка при програмиране 8.9

Грешка при компилация възниква, когато се прави опит за изпълнение на импортиран `static` метод със същото име и подпис като това на метод от същия клас или `static` импорт библиотека.

8.12 `final` данни на обект

- Принцип на “най- малката привилегия”
 - Кодът на програмата трябва да има само тези права за достъп, които позволяват да се изпълни заданието, но не и повече
- `final` данни на обект
 - Използват ключовата дума `final`
 - Декларира константа (**readonly**)
 - `final` данни на обект могат да се инициализират след декларацията им
 - Най- правилно е да се инициализират в конструктора
 - Всеки обект има свое копие на тази константа, в общия случай с различна стойност

Software Engineering факти 8.13

Деклариране на променлива на обект като final спомага за прилагане на принципа за най- малката привилегия.

Това е вярно в случай, че дадена променлива не трябва да се променя, след като веднъж е инициализирана- това води до избягване на вероятно грешна повторна инициализация.

Outline

Increment.java

```

1  // Fig. 8.15: Increment.java
2  // final instance variable in a class.
3
4  public class Increment
5  {
6      private int total = 0; // total of all increments
7      private final int INCREMENT; // constant variable (uninitialized)
8
9      // constructor initializes final instance variable INCREMENT
10     public Increment( int incrementValue )
11     {
12         INCREMENT = incrementValue; // initialize constant variable
13     } // end Increment constructor
14
15     // add INCREMENT to total
16     public void addIncrementToTotal()
17     {
18         total += INCREMENT;
19     } // end method addIncrementToTotal
20
21     // return String representation of an Increment object's data
22     public String toString()
23     {
24         return String.format( "total = %d", total );
25     } // end method toString
26 } // end class Increment

```

Декларира **final**
данни на
обекта this

Инициализира **final**
променлива на обект в
конструктора!



Outline

IncrementTest.java

```
1 // Fig. 8.16: IncrementTest.java
2 // final variable initialized with a constructor argument.
3
4 public class IncrementTest
5 {
6     public static void main( String args[] )
7     {
8         Increment value = new Increment( 5 );
9
10        System.out.printf( "Before incrementing: %s\n\n", value );
11
12        for ( int i = 1; i <= 3; i++ )
13        {
14            value.addIncrementToTotal();
15            System.out.printf( "After increment %d: %s\n", i, value );
16        } // end for
17    } // end main
18 } // end class IncrementTest
```

Създава **Increment** обект

Извиква метод
addIncrementToTotal()

Before incrementing: total = 0

After increment 1: total = 5

After increment 2: total = 10

After increment 3: total = 15



Обичайна грешка при програмиране 8.10

Attempting to modify a final instance variable after it is initialized is a compilation error.

Software Engineering факт 8.14

Една **final** данна трябва да се декларира **static**, ако тя ще **има една и съща константна стойност за всички обекти от даден клас.**

Тогава тази константа трябва да се инициализира на мястото на декларацията си

Обичайна грешка при програмиране 8.11

Пропускане на инициализация на *final* данна води до грешка при компилация.

Outline

Increment.java

```
Increment.java:13: variable INCREMENT might not have been initialized
    } // end Increment constructor
    ^
1 error
```



8.13 Изброим тип данни

- `enum` ключова дума за тип данни
- **Спадат към референтен** тип- обектите им се реферират с променливи
- Служи за дефиниране на списък от константи, представени общо с уникални, лесно читаеми имена
 - **Декларират** се с `enum` ключова дума
 - Списък от константи разделени със запетая след `enum`
 - Всяка константа има **име**, а може и да има **аргументи**
 - Декларират `enum class` със следните ограничения:
 - `enum` е тип, който неявно е **final**
 - `enum` константите са също **static**
 - Опит за създаване на обект от `enum` тип посредством **new** дава **грешка** при компилация
 - `enum` константи се използват **като всички други константи**
 - `enum` **конструктор**
 - Може да има допълнителни дефиниции т.е. да се използва като конструктор за общо ползване

8.13 Изброим тип данни...

- Най- прост вид- служат за дефиниране на група от константи.

```
enum Status{ CONTINUE, WON, LOST}; // static!
```

```
// .... in some method
```

```
Status mode = Status.WON;
```

```
// ...change it
```

```
mode = Status.LOST;
```

8.13 Изброим тип данни...

- Най- прост вид- служат за дефиниране на група от константи.

```
// class data member
enum Status{ CONTINUE, WON, LOST};
void someMethod()
{
    // ...
    Status mode = Status.WON;
    // ...
    mode = Status.LOST;
}
```

Важно: *enum* константите не могат да са локални!

8.13 Изброим тип данни...

- Могат да се използват със `switch()`

```
enum Answer{ OK, NO};  
boolean  someTest(Answer ask)  
{  
    switch (ask)  
    {  
        case OK :  
            return true;  
        case NO :  
            return false;  
    }  
    return false;  
}
```

Outline

Book.java

(1 от 2)

```

1 // Fig. 8.10: Book.java
2 // Declaring an enum type with constructor and explicit instance fields
3 // and accessors for these field
4
5 public enum Book
6 {
7     // declare constants of enum type
8     JHTP6( "Java How to Program 6e", "2005" ),
9     CHTP4( "C How to Program 4e", "2004" ),
10    IW3HTP3( "Internet & World wide Web How to Program 3e", "2004" ),
11    CPPHTP4( "C++ How to Program 4e", "2003" ),
12    VBHTP2( "Visual Basic .NET How to Program 2e", "2002" ),
13    CSHARPHTP( "C# How to Program", "2002" );
14
15    // instance fields
16    private final String title; // book title
17    private final String copyrightYear; // copyright year
18
19    // enum constructor
20    Book( String bookTitle, String year )
21    {
22        title = bookTitle;
23        copyrightYear = year;
24    } // end enum Book constructor
25

```

Декларира 6 **enum**
константи

Имена на **enum**
константи

Структура от
аргументи на **enum**
константи, които се
предават на
конструктора на
enum

Декларираме клас данни, съответстват
по брой и тип на аргументите на
константите

Дефинира конструктора за
enum Book



Outline

Book.java

(2 от 2)

```
26 // accessor for field title
27 public String getTitle()
28 {
29     return title;
30 } // end method getTitle
31
32 // accessor for field copyrightYear
33 public String getCopyrightYear()
34 {
35     return copyrightYear;
36 } // end method getCopyrightYear
37 } // end enum Book
```



8.13 Изброим тип данни...

- Всяка *enum* константа е обект от тип *Book* и има свое копие от клас данните *title* and *copyrightYear* (константни и статични!)
- Конструкторът (редове 20- 24) взима два *String* параметъра, първият задава заглавието на книга, а вторият, авторските права на книгата.
- Редове 22- 23 присвояват тези променливи на клас данните.
- Редове 27-36 декларират GET методи за *title* и *copyrightYear*
- *enum* типът позволява да обхождаме константите.

8.13 Изброим тип данни...

- *static* метод *values()*
 - Генерира се от компилаторът за всеки `enum` тип
 - Връща масив an array of the `enum`'s constants in the order in which they were declared
- *static* метод *range()* от *class EnumSet*
 - Взима два аргумента- **първия** и **последния** в желан **интервал** от *enum* константи
 - Връща *EnumSet* съдържащ константите в желания интервал, включително двата крайни обекта
 - Специализирана *for* кованда може да обходи множество *EnumSet* също както масив

Outline

EnumTest.java

(1 от 2)

```

1 // Fig. 8.11: EnumTest.java
2 // Testing enum type Book.
3 import java.util.EnumSet;
4
5 public class EnumTest
6 {
7     public static void main( String args[] )
8     {
9         System.out.println( "All books:\n" );
10
11         // print all books in enum Book
12         for ( Book book : Book.values() )
13             System.out.printf( "%-10s%-45s%\n", book,
14                               book.getTitle(), book.getCopyrightYear() );
15
16         System.out.println( "\nDisplay a range of enum constants:\n" );
17
18         // print first four books
19         for ( Book book : EnumSet.range( Book.JHTP6, Book.CPPHTP4 ) )
20             System.out.printf( "%-10s%-45s%\n", book,
21                               book.getTitle(), book.getCopyrightYear() );
22     } // end main
23 } // end class EnumTest

```

Специализиран **for** цикъл обхожда всяка от **enum** константите в **масива** върнат от метод **value()**

Връща името на текущата **enum** константа

Специализиран **for** цикъл обхожда всяка от **enum** константите **МНОЖЕСТВОТО** **EnumSet** върнато от метод **range**

Извежда **title** и **copyrightYear** на **текущо обработваната enum** **КОНСТАНТА**



Outline

EnumTest.java

(2 от 2)

All books:

JHTP6	Java How to Program 6e	2005
CHTP4	C How to Program 4e	2004
IW3HTP3	Internet & World Wide Web How to Program 3e	2004
CPPHTP4	C++ How to Program 4e	2003
VBHTP2	Visual Basic .NET How to Program 2e	2002
CSHARPHTP	C# How to Program	2002

Display a range of enum constants:

JHTP6	Java How to Program 6e	2005
CHTP4	C How to Program 4e	2004
IW3HTP3	Internet & World Wide Web How to Program 3e	2004
CPPHTP4	C++ How to Program 4e	2003



Обичайна грешка при програмиране 8.6

Синтактична грешка при *enum* декларация е да се декларира *enum* константите след *enum* конструкторите, данните или методите в *enum* декларацията.

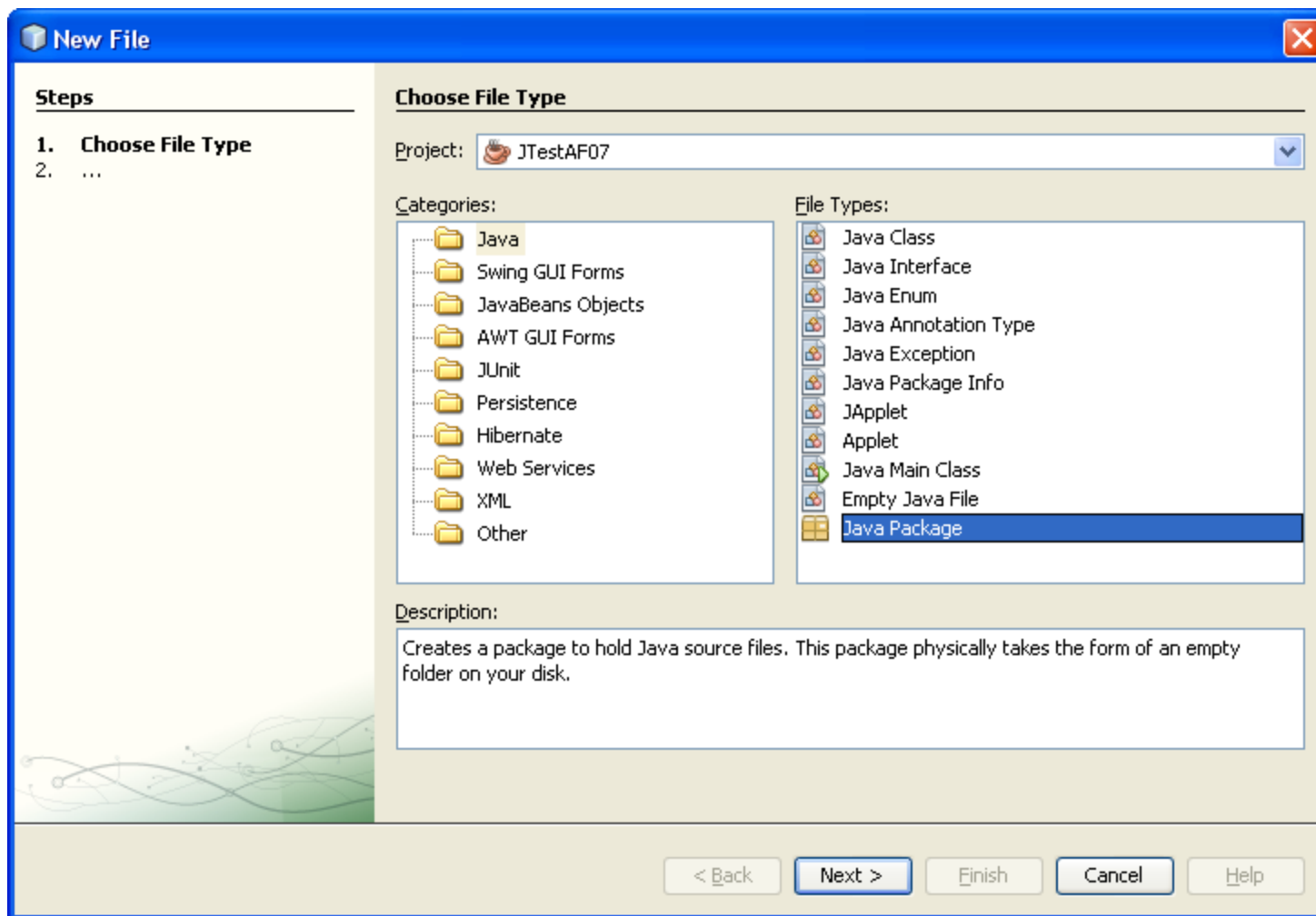
8.14 Повторно използване на код

- Бърза разработка на приложения
 - Ускорява разработката на високо качествени софтуерни приложения
- За многократна употреба на един или повече класове те се групират в библиотеки, наречани “*пакет*” (package) в термините на Java
- Java’s API
 - Множество от библиотеки на Java
 - Документация:
 - java.sun.com/j2se/5.0/docs/api/index.html
 - или java.sun.com/j2se/5.0/download.html

8.14 Повторно използване на код

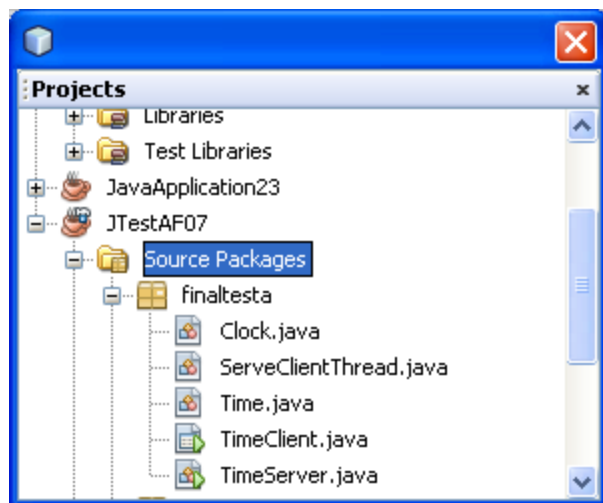
Създаване то на “*пакет*” започва с избиране на име на пакета и декларирането му с ключовата дума `package`

Създаване на име на нов Java package | NetBeans



8.14 Повторно използване на код

Впоследствие, към този “*пакет*” се добавят класовете, които изграждат package библиотеката. Например, класовете `Clock.java`, `ServerClientThread.java`, ... и пр. са добавени към `package finaltesta`.



8.15 Time Class пример: създаване на библиотеки (package)

- Декларират клас за повторна употреба
 - Добавяме package декларация **преди всички други *import* декларации**
 - package името е обикновено Internet домейн името написано в обратен ред и следвано от други имена, задаващи името на библиотеката
 - пример: **com.deitel.jhttp6.ch08**
 - package името се използва за арефериране на имената на класове в тази библиотека (package)
 - Във всяка библиотека (package) имената на класовете са различни
 - Предпазва дублиране, колизия на имена
 - Името на клас без да се използва package името се нарича просто име на клас

8.15 Time Class пример: създаване на библиотеки (package)

Стъпките за създаване на повторно използван клас са:

- 1. **Деклариране на *public class***. Ако класа не е *public*, той ще може да се **използва само** от класовете на същата библиотека (package), където е деклариран.
- 2. Изберете package име и добавете package декларация в сорс файла на класа за многократна употреба. Може да има само една декларация на package във всеки отделен сорс файл и тя трябва **да е първа** в началото на файла.
- 3. Компилирайте сорс файла, при което съответни *.class* файл се записва (автоматично) в поддиректория с името на библиотеката (package).
Например: *.class* файла ще се намира в директория *my/lib* при *package my.lib;*
- 4. Импортирайте *package* клас-а с *import* декларация и използвайте този клас все едно е дефиниран в същия сорс файл.

8.15 Time Class пример: създаване на библиотеки (package)

Всеки Java source-файл трябва да има следната структура :

1. Декларация на ***package*** (ако е нужно),
2. ***import*** декларации (ако са нужни), следвани от
3. ***class*** декларации (поне един ***public*** клас), .

Outline

```

1 // Fig. 8.18: Time1.java
2 // Time1 class declaration maintains the time in 24-hour format.
3 package com.deitel.jhttp6.ch08;
4
5 public class Time1
6 {
7     private int hour;    // 0 - 23
8     private int minute; // 0 - 59
9     private int second; // 0 - 59
10
11     // set a new time value using universal time; perform
12     // validity checks on the data; set invalid values to zero
13     public void setTime( int h, int m, int s )
14     {
15         hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); // validate hour
16         minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); // validate minute
17         second = ( ( s >= 0 && s < 60 ) ? s : 0 ); // validate second
18     } // end method setTime
19

```

package декларация. **Time1.class** ще се намира в директория **com/deitel/jhttp6/ch08** след компилация

Time1 е **public** class и може да се използва от класове в други библиотеки

Time1.java

(1 от 2)




```
20 // convert to String in universal-time format (HH:MM:SS)
21 public String toUniversalString()
22 {
23     return String.format( "%02d:%02d:%02d", hour, minute, second );
24 } // end method toUniversalString
25
26 // convert to String in standard-time format (H:MM:SS AM or PM)
27 public String toString()
28 {
29     return String.format( "%d:%02d:%02d %s",
30         ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
31         minute, second, ( hour < 12 ? "AM" : "PM" ) );
32 } // end method toString
33 } // end class Time1
```

Outline

Time1.java

(2 от 2)



8.15 Time Class пример: ...

- Компилираме сорс файла

- Пример: всички **.class** файлове на нашият **package** ще се намират в директория

```
com
├── deitel
│   └── jhttp6
│       └── ch08
```

- **javac** команда с опция **-d**
 - Създава необходимите директории съответстващи на **package** декларацията
 - Използване на **(.)** след опцията **-d** означава създаване на директории относно текущата директория

8.15 Time Class пример: ...

– Импортиране на `package class` в програма (друг клас)

- Пример за **импорт на отделен** `package` клас

```
import java.util.Random;
```

- Пример за **импорт на всички класове** от даден `package`

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

Outline

```
1 // Fig. 8.19: Time1PackageTest.java
2 // Time1 object used in an application.
3 import com.deitel.jhtp6.ch08.Time1; // import class Time1
4
5 public class Time1PackageTest
6 {
7     public static void main( String args[] )
8     {
9         // create and initialize a Time1 object
10        Time1 time = new Time1(); // calls Time1 constructor
11
12        // output string representations of the time
13        System.out.print( "The initial universal time is: " );
14        System.out.println( time.toUniversalString() );
15        System.out.print( "The initial standard time is: " );
16        System.out.println( time.toString() );
17        System.out.println(); // output a blank line
18    }
```

import декларация за отделен клас

Time1PackageTest
.java

(1 от 2)

Рефериране на **Time1**
class по име както
ако беше дефиниран
в същия сорс файл
или package



Outline

Time1PackageTest .java

(2 of 2)

```
19 // change time and output updated time
20 time.setTime( 13, 27, 6 );
21 System.out.print( "Universal time after setTime is: " );
22 System.out.println( time.toUniversalString() );
23 System.out.print( "Standard time after setTime is: " );
24 System.out.println( time.toString() );
25 System.out.println(); // output a blank line
26
27 // set time with invalid values; output updated time
28 time.setTime( 99, 99, 99 );
29 System.out.println( "After attempting invalid settings:" );
30 System.out.print( "Universal time: " );
31 System.out.println( time.toUniversalString() );
32 System.out.print( "Standard time: " );
33 System.out.println( time.toString() );
34 } // end main
35 } // end class Time1PackageTest
```

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```



8.15 Time Class пример: ...

- **Class loader** (зареждане на клас в оперативната памет)
 - Намира класовете които се използват от компилатора
 - Първо се търси стандартните библиотеки на JDK
 - Накрая търси package дефинирани с *classpath*
 - Задава списък директории или архивирани файлове
 - Архивираните файлове са `.jar` или `.zip`
 - Стандартните библиотеки са архивирани във файлове с окончания `.rt` или `.jar`

8.15 Time Class пример: ...

- **classpath** се използва за указване на директории различни от текущата
- Задава се чрез
 - **classpath** опция на **javac** компилатора
 - CLASSPATH** environment променлива на ОС
- **JVM** също трябва да намери **.class** файловете, както и компилатора
 - **java** командата използва за цялата същата опция както и **javac** командата

8.16 Package достъп

Package достъп

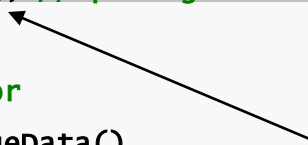
- По подразбиране класове и членовете им без явно указани права за достъп (`public` или `private`) са достъпни само в рамките на `package`-а където са декларирани
- Няма ефект при програма само с един клас
- Има значение при използване на библиотеки с повече от един клас

Outline

PackageDataTest .java

(1 от 2)

```
23 // class with package access instance variables
24 class PackageData
25 {
26     int number; // package-access instance variable
27     String string; // package-access instance variable
28
29     // constructor
30     public PackageData()
31     {
32         number = 0;
33         string = "Hello";
34     } // end PackageData constructor
35
36     // return PackageData object String representation
37     public String toString()
38     {
39         return String.format( "number: %d; string: %s", number, string );
40     } // end method toString
41 } // end class PackageData
```



Данни с package достъп

After instantiation:
number: 0; string: Hello

After changing values:
number: 77; string: Goodbye



Outline

PackageDataTest
.java

(2 от 2)

```
1 // Fig. 8.20: PackageDataTest.java
2 // Package-access members of a class are accessible by other classes
3 // in the same package.
4
5 public class PackageDataTest
6 {
7     public static void main( String args[] )
8     {
9         PackageData packageData = new PackageData();
10
11         // output String representation of packageData
12         System.out.printf( "After instantiation:\n%s\n", packageData );
13
14         // change package access data in packageData object
15         packageData.number = 77;
16         packageData.string = "Goodbye";
17
18         // output String representation of packageData
19         System.out.printf( "\nAfter changing values:\n%s\n", packageData );
20     } // end main
21 } // end class PackageDataTest
22
```

Директен достъп до всички достъпни
членове

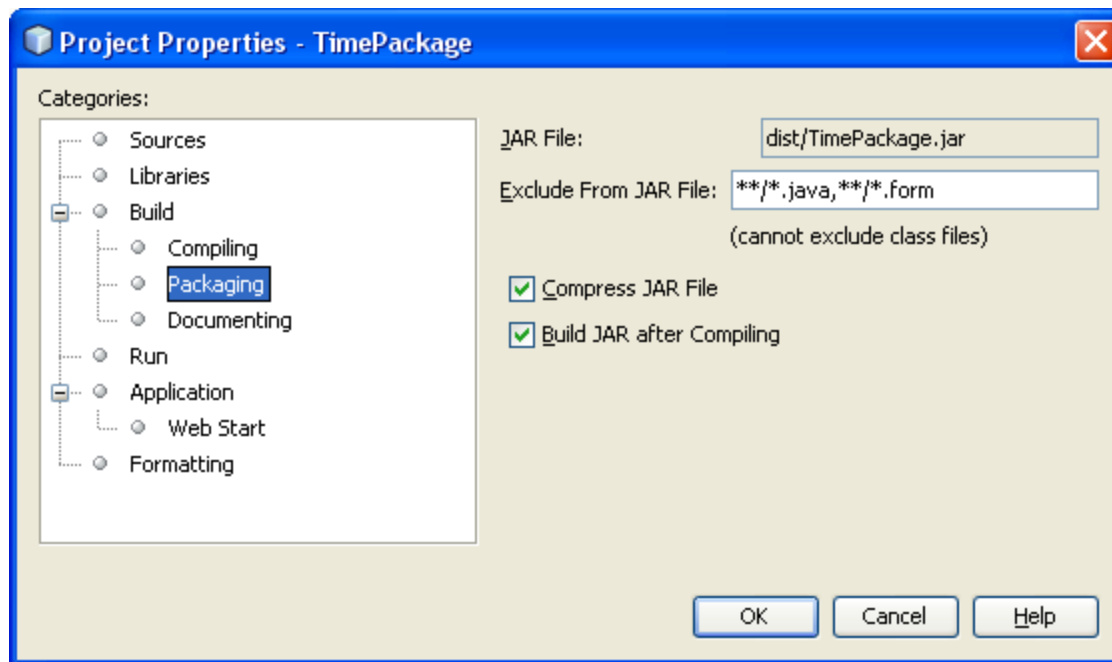


8.16 Package достъп

При многократно използване на package съдържанието му обикновено се архивира в JAR файл по следния начин:

- Изберете *File -> "<YouAppName>" Properties*
- В '*Categories*' групата изберете '*Packaging*' раздела, за да **видите името и директорията**, където NetBeans създава JAR файла. По подразбиране това е *dist/<ProjectName>.jar*
- Евентуално, изключете добавяне на файлове в JAR файл като използвате "*Exclude From JAR File*"
- Изберете '*Compress JAR File*' и '*Build JAR After Compiling*' и накрая потвърдете с *Press 'OK'*
- **Компилирайте** проекта си.
- **Дайте копие от създадения** JAR файл от *<Your Project Path>/dist/* на потребителите на този пакет (package)

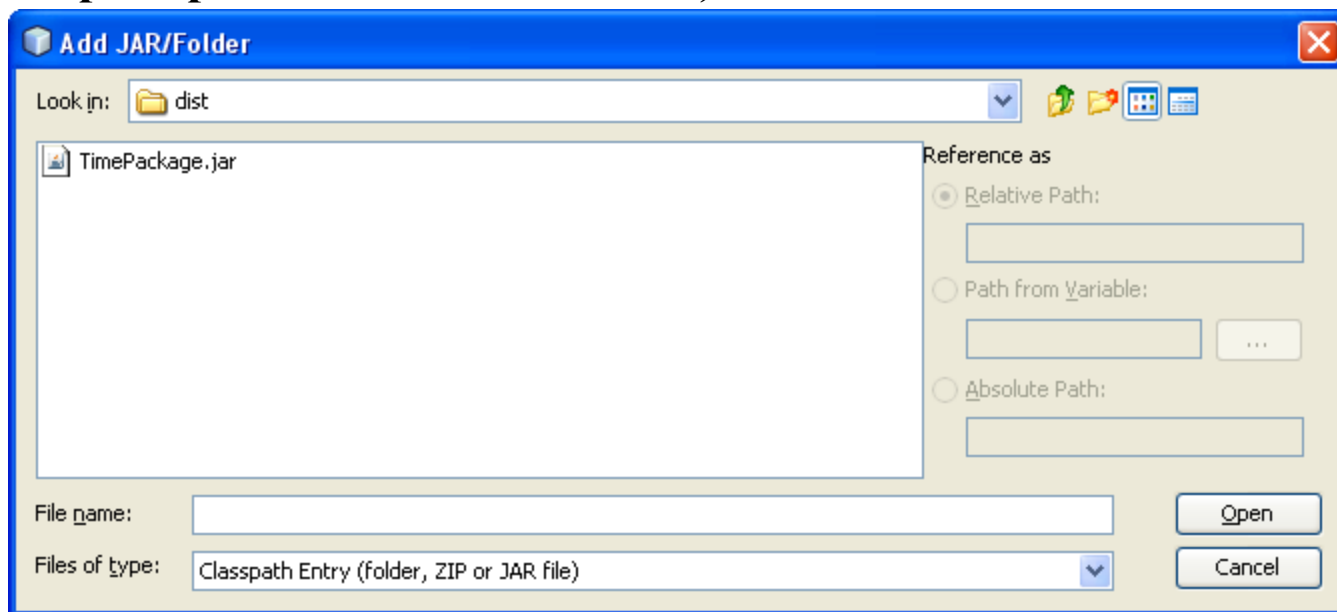
8.16 Package достъп



8.16 Package достъп

За използване на Jar файл с NetBeans:

1. Кликнете с десния бутон на мишката върху възела **Libraries** в **Projects** изгледа на проекта и изберете **Add Jar/Folder**
2. Изберете съществуващ **Jar** файл от директория и натиснете **Open**
3. Добавете **import** към името на **package** и клас , който ще се използва в новия проект (виж **TimePackage** и **PackageTest** проектите от примерния код към лекцията)



8.17 GUI и Graphics пример: Използване на обекти с Graphics

- Дефиниране на метода за рисуване на обект към дефиницията на обекта
 - Всеки път този обект се рисува по същия начин при извикване метод `paintComponent()`

Outline

myline.java

```

1 // Fig. 8.21: MyLine.java
2 // Declaration of class MyLine.
3 import java.awt.Color;
4 import java.awt.Graphics;
5
6 public class MyLine
7 {
8     private int x1; // x coordinate of first endpoint
9     private int y1; // y coordinate of first endpoint
10    private int x2; // x coordinate of second endpoint
11    private int y2; // y coordinate of second endpoint
12    private Color myColor; // color of this shape
13
14    // constructor with input values
15    public MyLine( int x1, int y1, int x2, int y2, color color )
16    {
17        this.x1 = x1; // set x coordinate of first endpoint
18        this.y1 = y1; // set y coordinate of first endpoint
19        this.x2 = x2; // set x coordinate of second endpoint
20        this.y2 = y2; // set y coordinate of second endpoint
21        myColor = color; // set the color
22    } // end MyLine constructor
23
24    // Draw the line in the specified color
25    public void draw( Graphics g )
26    {
27        g.setColor( myColor );
28        g.drawLine( x1, y1, x2, y2 );
29    } // end method draw
30 } // end class MyLine

```

Декларираме данни на обектите- координати и цвят

Инициализация на данните на обекта

Рисуване на линия при текущите координати и цвят



Outline

DrawPanel.java

(1 от 2)

```
1 // Fig. 8.22: DrawPanel.java
2 // Program that uses class MyLine
3 // to draw random lines.
4 import java.awt.Color;
5 import java.awt.Graphics;
6 import java.util.Random;
7 import javax.swing.JPanel;
8
9 public class DrawPanel extends JPanel
10 {
11     private Random randomNumbers = new Random();
12     private MyLine lines[]; // array of lines
13
14     // constructor, creates a panel with random shapes
15     public DrawPanel()
16     {
17         setBackground( Color.WHITE );
18
19         lines = new MyLine[ 5 + randomNumbers.nextInt( 5 ) ];
20     }
```

Деклариране на масив
от линии **MyLine**

Създаване на масива **MyLine**



Outline

DrawPanel.java

(2 от 2)

Генериране на координатите на всяка линия

Генериране на цвят

Създаване на обекти линии от клас
MyLine

Рисувани на всяка линия **MyLine**.
Обектът **Graphics** се задава от
JVM

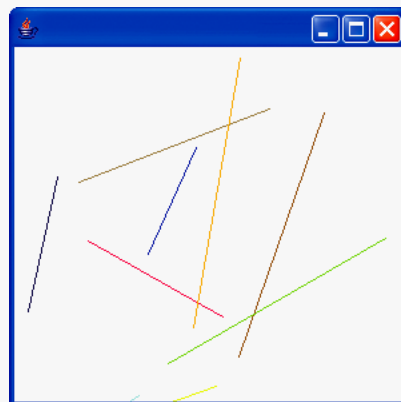
```
21 // create lines
22 for ( int count = 0; count < lines.length; count++ )
23 {
24     // generate random coordinates
25     int x1 = randomNumbers.nextInt( 300 );
26     int y1 = randomNumbers.nextInt( 300 );
27     int x2 = randomNumbers.nextInt( 300 );
28     int y2 = randomNumbers.nextInt( 300 );
29
30     // generate a random color
31     Color color = new Color( randomNumbers.nextInt( 256 ),
32                             randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );
33
34     // add the line to the list of lines to be displayed
35     lines[ count ] = new MyLine( x1, y1, x2, y2, color );
36 } // end for
37 } // end DrawPanel constructor
38
39 // for each shape array, draw the individual shapes
40 public void paintComponent( Graphics g )
41 {
42     super.paintComponent( g );
43
44     // draw the lines
45     for ( MyLine line : lines )
46         line.draw( g );
47 } // end method paintComponent
48 } // end class DrawPanel
```



Outline

TestDraw.java

```
1 // Fig. 8.23: TestDraw.java
2 // Test application to display a DrawPanel.
3 import javax.swing.JFrame;
4
5 public class TestDraw
6 {
7     public static void main( String args[] )
8     {
9         DrawPanel panel = new DrawPanel();
10        JFrame application = new JFrame();
11
12        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13        application.add( panel );
14        application.setSize( 300, 300 );
15        application.setVisible( true );
16    } // end main
17 } // end class TestDraw
```



Задачи

1. Напишете следните класове като спазвате концепциите за скриване на информация (*encapsulation, information hiding*) и изискване за многократно използване на код (*software reuse-избягване на дублиране на код!*)
 - Напишете *class Point* . Нека обектите на *class Point* имат променливи *x* и *y* (цели числа), характеризиращи координатите на съответната точка.
 - Дефинирайте за *class Point* пълен набор от конструктори за този клас (по подразбиране, за общо ползване и копирани), *set* и *get методи* за данните на класа, а също и *toString()* метод.

Задачи

- *Напишете* също *class Rectangle*, чиито обекти имат две данни-*uPoint* и *lPoint*, които са референции към обекти от *class Point* и задават горния ляв ъгъл и долния десен ъгъл на правоъгълника.
- *Дефинирайте* за *class Rectangle* пълен набор от *конструктори* за този клас (по подразбиране, за общо ползване и копиране), *set* и *get методи* за данните на класа, а също и *toString()* *метод*. (използвайте метода *toString()* дефиниран за обектите от *class Point*)
- *Напишете* също метод *draw(Graphics g)* в *class Rectangle* позволяващ да се рисува текущия *this* обект последством зададения като аргумент обект от клас *Graphics*.

Задачи

- *Напишете* също `class Line`, чиито обекти имат две данни-`sPoint` и `ePoint` `Points`, които са референции към обекти от `class Point` и задават началната и крайната точка на линията (отсечка).
- *Дефинирайте* за `class Line` пълен набор от *конструктори* за този клас (по подразбиране, за общо ползване и копирани), `set` и `get` *методи* за данните на класа, а също и `toString()` *метод*. (използвайте метода `toString()` дефиниран за обектите от `class Point`)
- *Напишете* също метод `draw(Graphics g)` в `class Line` позволяващ да се рисува текущия *this* обект последством зададения като аргумент обект от клас `Graphics`.

Задачи

- **Напишете също class *DrawTest* и class *DrawPanel* за тестване на тези класове:**
 - **Създаване на обекти от класове *Point*, *Rectangle* и *Line***
 - **Извеждане на информация за създадените обекти на стандартен изход, чрез *toString()***
 - **Рисуване на обектите *Rectangle* и *Line* в *JPanel*, чрез извикване на метода им *draw()* в метода *paintComponent(Graphics g)* на *DrawPanel***