

# Języki i metody programowania Java

## Lab4 – podejście obiektowe, zastosowanie pojemników

<https://docs.oracle.com/javase/tutorial/>

[http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/pojava/javazk4\\_2.pdf](http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/pojava/javazk4_2.pdf)

Zofia Kruczkiewicz

## **Zadanie 1. Zastosowanie klasy `HashSet` jako pojemnika na obiekty. Podział programu na pakiety wspierające budowę dwuwarstwowej aplikacji – wykonanie programu `Rys1_1`**

1. Należy wykonać projekt typu **Java Application** (wg instrukcji z Lab1)- należy wybrać **File/New Project/Java/Java Application** i wpisać nazwę projektu w polu **Project Name** i położenie w polu **Project Location** np. **Rys1\_1**
2. Należy w polu **Create Main Class** wpisać GUI. **Obraz1\_1.java**. W wyniku tych czynności powstanie pakiet **GUI** oraz klasa **Obraz1\_1.java**. Ta klasa będzie zawierała metodę `main` do uruchomienia całego programu, prezentując zaimplementowane algorytmy przetwarzania figur: tworzenie figur, prezentowanie danych figur, wyszukiwanie i zaznaczanie figur oraz rysowanie figur w trybie graficznym.
3. Należy wykonać kolejne pakiety o nazwach: **figury** i **grafika**. Sposób utworzenia pakietu: po kliknięciu prawym klawiszem myszy na nazwę projektu wybrać: **New/Other/Java/Java Package** i następnie klawisz **Next**. W polu **Package Name** należy wpisać nazwę pakietu i nacisnąć klawisz **Finish**.
4. W pakiecie **figury** należy wykonać dwie klasy: **Punkt** i **Kwadrat** (slajdy...). Sposób utworzenia klasy: po kliknięciu prawym klawiszem myszy na nazwę pakietu wybrać: **New/Other/Java/Java Class** i następnie klawisz **Next**. W polu **Class Name** należy wpisać nazwę klasy (i upewnić się, czy w polu **Package** jest wybrany właściwy pakiet – jeśli nie, można wybrać właściwy pakiet z listy pakietów pola **Package**) i nacisnąć klawisz **Finish**. Obiekty tych klas są przetwarzanymi figurami
5. W pakiecie **grafika** należy wykonać trzy klasy: **Figury**, **FiguryHashSet** i **Figury\_panel** (slajdy...), tak, jak w p.4. Obiekty tych klas służą do przetwarzania figur z pakietu **grafika**. Klasa **Figury** będzie zawierać uniwersalny kod, niezależny od typów pojemników implementujących interfejs **Collection**. Klasa **FiguryHashSet** będzie zawierać kod zależny od wybranej implementacji **HashSet**. Klasa **Figury\_panel** będzie zawierać kod umożliwiający prezentację figur w trybie graficznym z pakietu **figury**, przetwarzanych przez obiekty klas **Figury** i **FiguryHashSet**.
6. **Należy dodać własną figurę: koło, trójkąt lub prostokąt zachowując zasady polimorfizmu i dodać metodę `lezy_na` wg slajdu 8 w celu wprowadzenia hermetyzacji do metody `Clicked`.**

## Pakiet **figury**: w pliku **Punkt.java** należy wkleić kod klasy podany poniżej

```
package figury;  
  
import java.awt.Color;  
import java.awt.Graphics;  
import java.awt.Graphics2D;  
public class Punkt implements Comparable {
```

```
    protected int x, y;  
    public Punkt(int wspX, int wspY) {  
        x = wspX;  
        y = wspY; }  
  
    public int getX()        { return x; }  
    public int getY()        { return y; }
```

```
    @Override  
    public int compareTo(Object o) {  
        Punkt p = (Punkt) o;  
        if ((x == p.x) && (y == p.y)) {  
            return 0;  
        } else if ((x < p.x) && (y < p.y)) {  
            return -1; }  
        return 1; }  
  
    @Override  
    public int hashCode() {
```

```
        int hash = 7;  
        hash = 17 * hash + this.x;  
        hash = 17 * hash + this.y;  
        return hash; }
```

Metoda **compareTo** implementuje metodę interfejsu **Comparable**, przygotowując kod klasy **Punkt** i jej pochodnych do umieszczenia w pojemnikach sortujących np **TreeSet**, **TreeMap**.

Kod metody **hashCode** można wygenerować za pomocą opcji **Insert Code...** Metoda ta potrzebna jest obiektom typu **Punkt**, jeśli są umieszczane w pojemnikach implementujących interfejs **Collection**

@Override

```
public boolean equals(Object obj) {  
    if (this == obj)           { return true; }  
    if (obj == null)           { return false; }  
    if (getClass() != obj.getClass()) { return false; }  
    return this.compareTo(obj)==0;  
}
```

Kod metody **equals** można wygenerować za pomocą opcji **Insert Code...** i część kodu zmodyfikować za pomocą wykorzystania metody **compareTo**. Metoda **equals** potrzebna jest obiektom typu **Punkt**, jeśli są umieszczane w pojemnikach implementujących interfejs **Collection**

```
public double odleglosc(Punkt p) {  
    return Math.sqrt((x - p.x) * (x - p.x) + (y - p.y) * (y - p.y)); }  
}
```

Metoda **odleglosc** – wyjaśniona w instrukcji do lab2

@Override

```
public String toString() {  
    return "Punkt{" + "x=" + x + ", y=" + y + "}"; }  
}
```

Kod metody **toString** można wygenerować za pomocą opcji **Insert Code...**

```
public int getDI() { return 5; }  
}
```

Kod metody **getDI** można wygenerować za pomocą opcji **Insert Code...** Służy ona określenia rozmiaru obiektu typu **Punkt** prezentowanego np na ekranie w postaci koła

```
public void przesun(int dx, int dy, int a, int b) {  
    x += dx;  
    y += dy;  
    if (x > a || x < 1) { x = 5; }  
    if (y > b || y < 1) { y = 2; }  
}
```

Kod metody **przesun** będzie zastosowany do przesuwania obiektu typu **Punkt** na ekranie w trybie graficznym w obszarze (0,0..a,b)

```
public void rysuj(Graphics g) {  
    Graphics2D g2D = (Graphics2D) g;  
    Color pedzel = new Color(255, 0, 0);  
    g2D.setColor(pedzel);  
    g2D.fillOval(x, y, 5, 5);  
}
```

Kod metody **rysuj** będzie zastosowany do prezentowania obiektu typu **Punkt** w trybie graficznym

}

Pakiet **figury**: pliku **Kwadrat.java** należy wkleić kod klasy podany poniżej

```
package figury;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;

public class Kwadrat extends Punkt {

    protected int dlugosc;

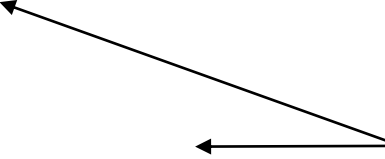
    public Kwadrat(int wspX, int wspY, int dlugosc_) {
        super(wspX, wspY);
        dlugosc = dlugosc_;
    }

    public int getDI() {
        return dlugosc;
    }

    public double odleglosc() {
        return Math.sqrt(x * x + y * y);
    }

    @Override
    public double odleglosc(Punkt p) {
        return odleglosc() + super.odleglosc(p);
    }
}
```

Metody przeciążone **odleglosc** –  
wyjaśnione w instrukcji do lab2



@Override

```
public int hashCode() {  
    int hash = 5;  
    hash = 97 * hash + this.dlugosc;  
    return hash;  
}
```

@Override

```
public boolean equals(Object obj) {  
    if (this == obj) { return true; }  
    if (obj == null) { return false; }  
    if (getClass() != obj.getClass()) { return false; }  
    final Kwadrat other = (Kwadrat) obj;  
    if (this.dlugosc != other.dlugosc) {  
        return false; }  
    return this.compareTo(obj)==0;  
}
```

@Override

```
public String toString() {  
    String s = super.toString();  
    return "Kwadrat{" + "dlugosc=" + dlugosc + "}' + " i dziedzicze od " + s;  
}
```

@Override

```
public void rysuj(Graphics g) {  
    Graphics2D g2D = (Graphics2D) g;  
    Color pedzel = new Color(0, 255, 0);  
    g2D.setColor(pedzel);  
    g2D.fillRect(x, y, dlugosc, dlugosc);  
}
```

```
}
```

Pakiet **grafika**: w pliku **Figury.java** należy wkleić kod klasy podany poniżej

```
package grafika;

import figury.Kwadrat;
import figury.Punkt;
import java.awt.Graphics;
import java.util.Collection;

public class Figury {

    protected int N = 3;
    public Collection<Punkt> figury;
    protected Punkt biezacy;

    public Punkt getBiezacy() {    return biezacy;    }

    public void polozenie() {
        for (Punkt figura : figury) {
            boolean p = figura instanceof Kwadrat;
            System.out.println(
                p + ", ze jestem kwadratem, bo jestem " + figura.toString()
                + ", X=" + figura.getX()
                + ", Y=" + figura.getY()
                + ", odleglosc=" + figura.odleglosc(figura));
        }
    }
}
```

Kod klasy **Figury** niezależny od implementacji interfejsu **Collection**

Kod metody **polozenie** prezentujący w trybie konsolowym dane figur typu **Punkt** i **dziedziczących po klasie Punkt**. Metoda **odleglosc** dla obiektów typu **Punkt** zwraca zero, natomiast dla obiektów typu **Kwadrat** zwraca odległość lewego górnego wierzchołka od początku współrzędnych – dzięki **przedefiniowaniu metody odleglosc** w klasie **Kwadrat** (polimorfizm). Uniwersalność kodu wynika z polimorfizmu metod **odleglosc** i **toString** i dziedziczenia metod **getX** i **getY**.

```

public boolean Clicked(int x_, int y_)
{
    for (Punkt figura : figury) {
        if (figura.getX() + figura.getDI() >= x_
            && figura.getX() <= x_
            && figura.getY() + figura.getDI() >= y_
            && figura.getY() <= y_) {
            biezacy = figura;
            return true; }
        }
    return false;
}

public void rysuj_figury(Graphics g) {
    for (Punkt figura : figury) {
        figura.rysuj(g);
    }
}

public boolean przesun(int x, int y, int dl, int szer) {
    if (biezacy != null) {
        biezacy.przesun(x, y, dl, szer);
        return true; }
    return false;
}
}

```

Kod metody **Clicked**, sprawdzający, czy miejsce o współrzędnych  $x_$ ,  $y_$  jest położone na powierzchni figury. Kod tej metody powinien być poprawiony tak, aby sprawdzanie położenia tego miejsca wykonywały obiekty typu **Punkt** lub typu **Kwadrat**, znajdujące się w pojemniku **figury za pomocą metody `lezy_na`**, czyli:

```

public boolean Clicked(int x_, int y_) {
    for (Punkt figura : figury) {
        if(figura.lezy_na(x_, y_))
        { biezacy = figura;
          return true; }
        }
    return false;
}

```

**Jest to konieczne ze względu na hermytyzację przetwarzania składowych klas Punkt i Kwadrat.**



Pakiet **grafika**: w pliku **FiguryHashSet.java** należy wkleić kod klasy podany poniżej

```
package grafika;
```

```
import figury.Kwadrat;
```

```
import figury.Punkt;
```

```
import java.util.HashSet;
```

```
public class FiguryHashSet extends Figury {
```

```
    public void pojemnik() {
```

```
        figury = new HashSet();
```

```
    }
```

```
    public boolean wyszukaj(Punkt p) {
```

```
        return figury.contains(p); }
```

```
    public boolean wyszukaj(int dane[]) {
```

```
        Punkt p;
```

```
        if (dane[0] == 0) {
```

```
            p = new Punkt(dane[1], dane[2]);
```

```
        } else {
```

```
            p = new Kwadrat(dane[1], dane[2], dane[3]);
```

```
        }
```

```
        return wyszukaj(p);
```

```
    }
```

Kod klasy **FiguryHashSet** jest zależny od implementacji interfejsu **Collection** lub **Map**. Kolorem czerwonym zostały zaznaczone te zależne fragmenty kodu.

```
public void wypelnij() {  
    for (int i = 0; i < N; i++) {  
        figury.add(new Punkt(20 * (N+i), 10 * (N+i)));  
        figury.add(new Kwadrat((i + 1) * 20, (i + 1) * 20, 20));  
    }  
}
```

Sposób tworzenia elementu kolekcji zależy od typu pojemnika np elementy kolekcji **implementujących interfejs Map i implementujących interfejs Collection.**

```
public void init() {  
    pojemnik();  
    wypelnij();  
    polozenie();  
}  
}
```

Pakiet **grafika**: w pliku **Figury\_panel.java** należy wkleić kod klasy podany poniżej

```
package grafika;  
import java.awt.Graphics;  
import javax.swing.JPanel;
```

```
public class Figury_panel extends JPanel{  
  
FiguryHashSet kontroler;  
  
@Override  
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    kontroler.rysuj_figury(g);  
}  
  
    public void init()  
    { kontroler=new FiguryHashSet();  
      kontroler.pojemnik();  
      kontroler.wypelnij();  
    }  
}
```

Kod klasy **Figury\_panel** reprezentuje kod klasy pochodnej od **JPanel** do prezentowania w trybie graficznym wyników przetwarzania figur przez obiekt **kontroler** typu **FiguryHashSet** za pomocą metody **paintComponent**

Pakiet **GUI**: w pliku **Obraz1\_1.java** należy wkleić kod klasy podany poniżej

```
package GUI;

import grafika.FiguryHashSet;
import grafika.Figury_panel;
import javax.swing.JFrame;

public class Obraz1_1{

    void rysunek_Swing()
    {
        JFrame ramka = new JFrame();
        Figury_panel panel = new Figury_panel();
        panel.init();
        ramka.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        ramka.setSize(200, 200);
        ramka.setContentPane(panel);
        ramka.setVisible(true);
    }
}
```

Kod metody rysunek\_Swing, w którym tworzony jest obiekt typu JFrame, który obsługuje tryb graficzny:

- tworzy formularz (//1),
- tworzy obiekt pochodny od JPanel (czyli Figury\_panel //2)
- **pozwala zamknąć okno formularza i program za pomocą znaku X w prawym górnym rogu okna (//3)**
- **Ustawia rozmiary okna formularza - //4**
- zawartość tego formularza wypełnia obiektem **panel** pochodnym od JPanel (czyli typu Figury\_panel //2) - **//5**
- ustawienie stanu obiektu typu JFrame w tryb wyświetlania - **//6**

Pakiet **GUI**: w pliku **Obraz1\_1.java** należy wkleić kod klasy podany poniżej

```
void rysunek_konsola() {
    FiguryHashSet kontroler = new FiguryHashSet();
    kontroler.init();
    int dane[] = {0, 60, 30};
    int dane1[] = {1, 20, 20, 20};
    int dane2[] = {0, 1, 1};
    System.out.println("poszukanie punktu: " + kontroler.wyszukaj(dane));
    System.out.println("poszukanie kwadratu: " + kontroler.wyszukaj(dane1));
    System.out.println("poszukanie punktu: " + kontroler.wyszukaj(dane2));
    System.out.println("zaznaczono " + kontroler.Clicked(15, 15));
    System.out.println("zaznaczono " + kontroler.Clicked(60, 60));
    System.out.println("zaznaczono " + kontroler.Clicked(25, 25));
    System.out.println("przesunieto " + kontroler.przesun(15, 15, 100, 100));
    System.out.println(kontroler.getBiezacy());
    System.out.println("przesunieto " + kontroler.przesun(60, 60, 100, 60));
    System.out.println(kontroler.getBiezacy());
    System.out.println("przesunieto " + kontroler.przesun(25, 25, 100, 100));
}
public static void main(String args[]) {
    Obraz1_1 obraz = new Obraz1_1();
    obraz.rysunek_konsola(); //1
    obraz.rysunek_Swing(); //2
}
}
```

Kod metody **rysunek\_konsola** reprezentuje działanie metod obiektu kontroler typu **FiguryHashSet** przetwarzających kolekcję typu **HashSet** obiektów z rodziny **Punkt**.

Kod programu (metody main) prezentujący przetwarzanie figur w trybie

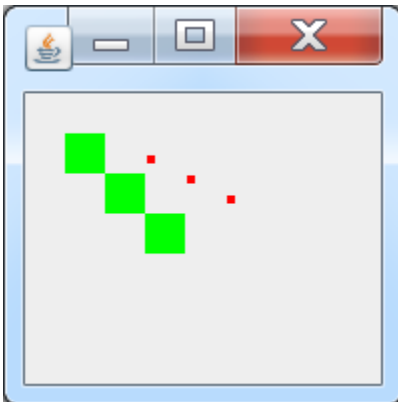
- konsolowym **//1**
- graficznym **//2**

## Uruchomienie programu **Rys1\_1** – należy zwrócić uwagę na kolejność wyświetlanych figur wynikającą z kolejności umieszczenia w pojemniku, czyli rodzaju pojemnika

run:

```
false, ze jestem kwadratem, bo jestem Punkt{x=60, y=30}, X=60, Y=30, odleglosc=0.0
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=20, y=20}, X=20, Y=20, odleglosc=28.284271247461902
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=40, y=40}, X=40, Y=40, odleglosc=56.568542494923804
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=60, y=60}, X=60, Y=60, odleglosc=84.8528137423857
false, ze jestem kwadratem, bo jestem Punkt{x=100, y=50}, X=100, Y=50, odleglosc=0.0
false, ze jestem kwadratem, bo jestem Punkt{x=80, y=40}, X=80, Y=40, odleglosc=0.0
poszukanie punktu: true
poszukanie kwadratu: true
poszukanie punktu: false
zaznaczono false
zaznaczono true
zaznaczono true
przesuniето true
Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=35, y=35}
przesuniето true
Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=95, y=2}
przesuniето true
```

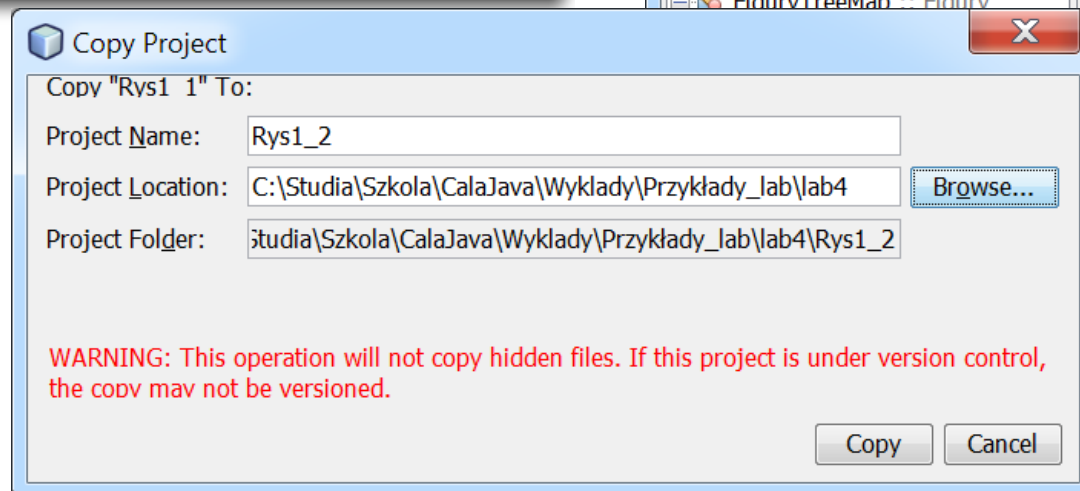
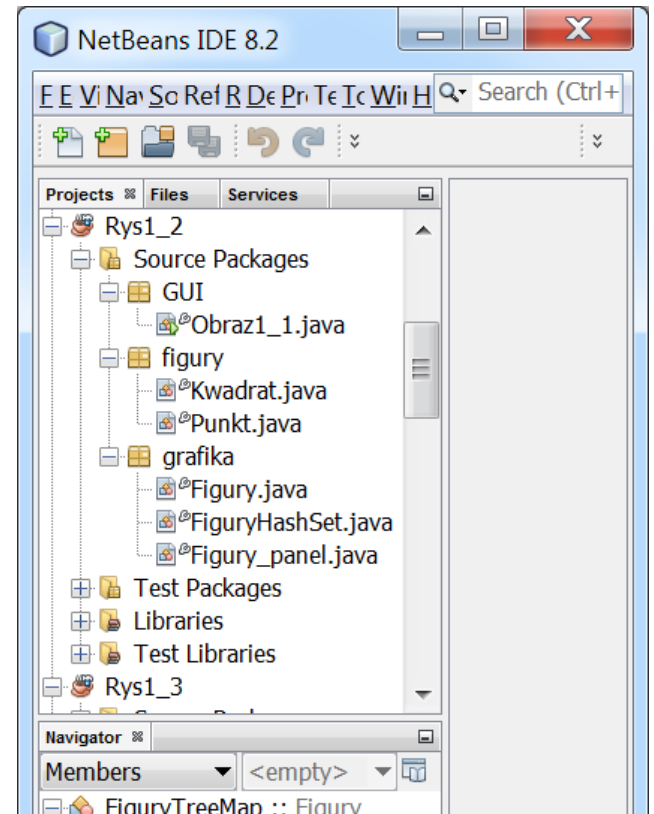
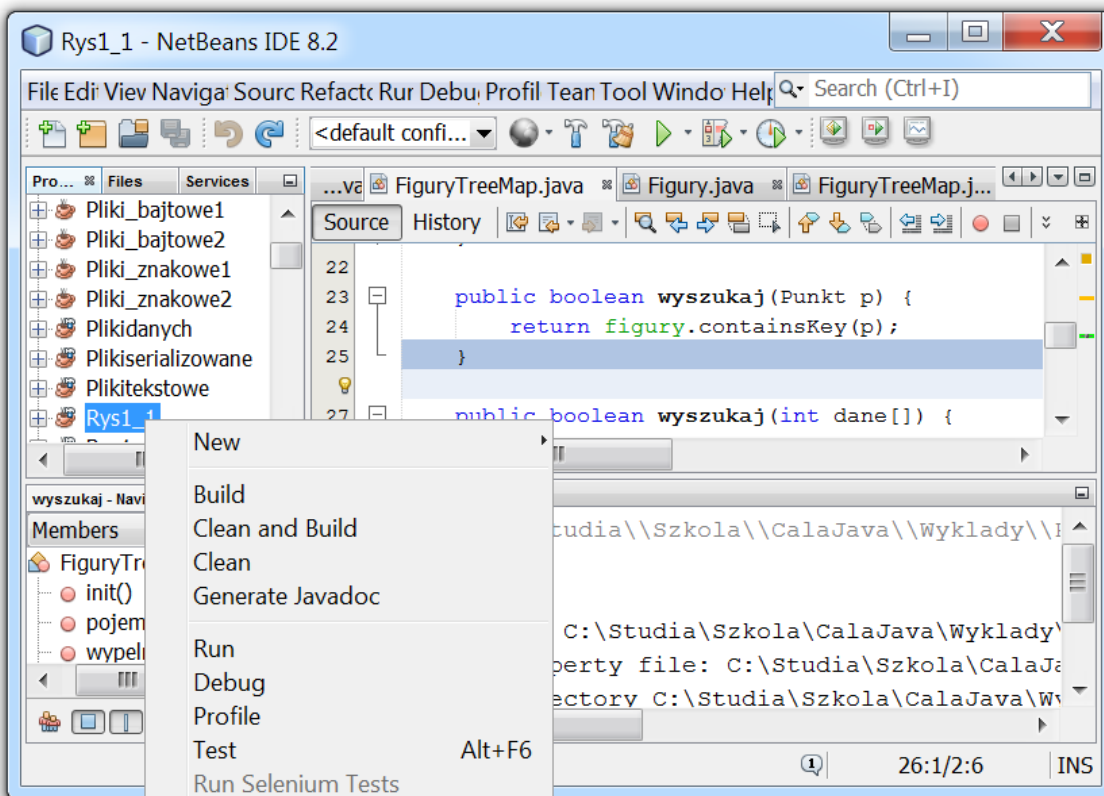
← **Wynik działania metody `rysunek_konsola`.** Zastosowany pojemnik typu **HashSet** wstawia elementy wg generowanej wartości przez metodę **hashCode** każdego elementu wstawianego do pojemnika. Obiekty były wstawiane na przemian: typu **Punkt** i typu **Kwadrat**



**Zadanie 2.** Zastosowanie klasy **ArrayList** jako pojemnika na obiekty. Podział programu na pakiety wspierające budowę dwuwarstwowej aplikacji – wykonanie programu **Rys1\_2**

1. Należy wykonać kopię projektu **Rys1\_1** jako **Rys1\_2** (slajd 16).
2. Należy zmodyfikować nazwy klas: **Obraz1\_1** na **Obraz1\_2** (pakiet GUI), **FiguryHashSet** na **FiguryArrayList** w pakiecie **grafika** wg slajdu 17
3. Należy zmodyfikować kod klas: **FiguryArrayList**, **Figury\_panel** oraz **Obraz1\_2** dostosowując je do nowego typu pojemnika **ArrayList**.
4. Należy dodać własną figurę: koło, trójkąt lub prostokąt zachowując zasady polimorfizmu i dodać metodę **lezy\_na** wg slajdu 8 w celu wprowadzenia hermetyzacji do metody **Clicked**.

# Ad.1 – Wykonanie kopii programu na przykładzie Rys1\_1 jako Rys1\_2





## Ad. 2 –Zmianay nazwy pliku jako przykład (z Obraz1\_1 na Obraz1\_2)

The image illustrates the process of renaming a class in NetBeans IDE 8.2. It shows two windows: 'Rys1\_2' and 'Rys1\_4'.

In the 'Rys1\_2' window, the 'Projects' view shows a project named 'Rys1\_2' with a package 'GUI' containing a file '@Obraz1\_1.java'. A context menu is open over this file, and the 'Refactor' option is selected. A dialog box titled 'Rename Class Obraz1\_1' is shown, with the 'New Name' field containing 'Obraz1\_2' and the checkbox 'Apply Rename on Comments' checked. The dialog has buttons for 'Preview', 'Refactor', 'Cancel', and 'Help'.

In the 'Rys1\_4' window, the 'Projects' view shows the same project 'Rys1\_2', but the file in the 'GUI' package is now '@Obraz1\_2.java'. The 'Output' window at the bottom right shows the command 'ant' and its output 'init' and 'deps'.

Ad3. Pakiet **grafika**: w pliku **FiguryArrayList.java** należy zmodyfikować kod klasy podany poniżej, zaznaczony na czerwono, zależny od zmiany typu pojemnika

```
package grafika;

import figury.Kwadrat;
import figury.Punkt;
import java.util.ArrayList;

public class FiguryArrayList extends Figury {

    public void pojemnik() {
        figury = new ArrayList();
    }
    public boolean wyszukaj(Punkt p) {
        return figury.contains(p); }

    public boolean wyszukaj(int dane[]) {
        Punkt p;
        if (dane[0] == 0) {
            p = new Punkt(dane[1], dane[2]);
        } else {
            p = new Kwadrat(dane[1], dane[2], dane[3]);
        }
        return wyszukaj(p);
    }
}
```

Ad3 cd: Pakiet **grafika**: w pliku **Figury\_panel.java** należy zmodyfikować kod klasy podany poniżej, zaznaczony na czerwono, zależny od zmiany typu pojemnika

```
package grafika;  
import java.awt.Graphics;  
import javax.swing.JPanel;
```

```
public class Figury_panel extends JPanel{
```

```
FiguryArrayList kontroler;
```

```
@Override
```

```
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    kontroler.rysuj_figury(g);  
}
```

```
public void init()
```

```
{ kontroler=new FiguryArrayList();  
    kontroler.pojemnik();  
    kontroler.wypelnij();  
}  
}
```

Ad3 cd: Pakiet **GUI**: w pliku **Obraz1\_1.java** należy zmodyfikować kod klasy podany poniżej, zaznaczony na czerwono, zależny od zmiany typu pojemnika (cd na kolejnym slajdzie)

```
package GUI;
```

```
import grafika.FiguryArrayList;
```

```
import grafika.Figury_panel;
```

```
import javax.swing.JFrame;
```

```
public class Obraz1_1{
```

```
    void rysunek_Swing()
```

```
    {
```

```
        JFrame ramka = new JFrame();
```

```
        Figury_panel panel = new Figury_panel();
```

```
        panel.init();
```

```
        ramka.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
        ramka.setSize(200, 200);
```

```
        ramka.setContentPane(panel);
```

```
        ramka.setVisible(true);
```

```
    }
```

Ad3 cd: Pakiet **GUI**: w pliku **Obraz1\_1.java** należy zmodyfikować kod klasy podany poniżej, zaznaczony na czerwono, zależny od zmiany typu pojemnika (cd poprzedniego slajdu)

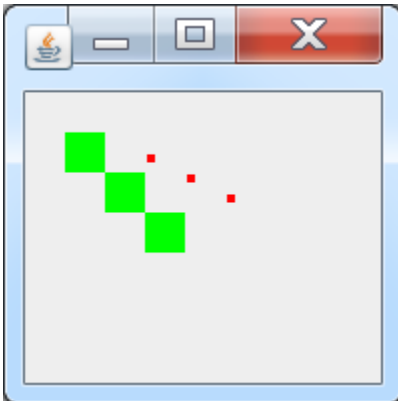
```
void rysunek_konsola() {  
    FiguryArrayList kontroler = new FiguryArrayList();  
    kontroler.init();  
    int dane[] = {0, 60, 30};  
    int dane1[] = {1, 20, 20, 20};  
    int dane2[] = {0, 1, 1};  
    System.out.println("poszukanie punktu: " + kontroler.wyszukaj(dane));  
    System.out.println("poszukanie kwadratu: " + kontroler.wyszukaj(dane1));  
    System.out.println("poszukanie punktu: " + kontroler.wyszukaj(dane2));  
    System.out.println("zaznaczono " + kontroler.Clicked(15, 15));  
    System.out.println("zaznaczono " + kontroler.Clicked(60, 60));  
    System.out.println("zaznaczono " + kontroler.Clicked(25, 25));  
    System.out.println("przesuniето " + kontroler.przesun(15, 15, 100, 100));  
    System.out.println(kontroler.getBiezacy());  
    System.out.println("przesuniето " + kontroler.przesun(60, 60, 100, 60));  
    System.out.println(kontroler.getBiezacy());  
    System.out.println("przesuniето " + kontroler.przesun(25, 25, 100, 100));  
}  
public static void main(String args[]) {  
    Obraz1_1 obraz = new Obraz1_1();  
    obraz.rysunek_konsola();  
    obraz.rysunek_Swing();  
}  
}
```

## Uruchomienie programu Rys1\_2 - należy zwrócić uwagę na kolejność wyświetlanych figur wynikającą z kolejności umieszczenia w pojemniku, czyli rodzaju pojemnika

run:

```
false, ze jestem kwadratem, bo jestem Punkt{x=60, y=30}, X=60, Y=30, odleglosc=0.0
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=20, y=20}, X=20, Y=20, odleglosc=28.284271247461902
false, ze jestem kwadratem, bo jestem Punkt{x=80, y=40}, X=80, Y=40, odleglosc=0.0
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=40, y=40}, X=40, Y=40, odleglosc=56.568542494923804
false, ze jestem kwadratem, bo jestem Punkt{x=100, y=50}, X=100, Y=50, odleglosc=0.0
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=60, y=60}, X=60, Y=60, odleglosc=84.8528137423857
poszukanie punktu: true
poszukanie kwadratu: true
poszukanie punktu: false
zaznaczono false
zaznaczono true
zaznaczono true
przesuniето true
Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=35, y=35}
przesuniето true
Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=95, y=2}
przesuniето true
```

← **Wynik działania metody `rysunek_konsola`.** Zastosowany pojemnik typu `ArrayList` wstawia elementy wg kolejności wstawiania obiektów typu `Punkt` i `Kwadrat`, czyli na przemian.



**Zadanie 3.** Zastosowanie klasy **TreeSet** jako pojemnika na obiekty. Podział programu na pakiety wspierające budowę dwuwarstwowej aplikacji – wykonanie programu **Rys1\_3**

1. Należy wykonać kopię projektu **Rys1\_1** jako **Rys1\_3** (podobnie jak na slajdzie 16).
2. Należy zmodyfikować nazwy klas: **Obraz1\_1** na **Obraz1\_3** (pakiet GUI), **FiguryHashSet** na **FiguryTreeSet** w pakiecie **grafika** wg slajdu 17
3. Należy zmodyfikować kod klas: **FiguryTreeSet**, **Figury\_panel** oraz **Obraz1\_3** dostosowując je do nowego typu pojemnika **TreeSet** podobnie jak w przypadku pojemnika **ArrayList**.
4. **Należy dodać własną figurę: koło, trójkąt lub prostokąt zachowując zasady polimorfizmu i dodać metodę lezy\_na wg slajdu 8 w celu wprowadzenia hermetyzacji do metody Clicked.**

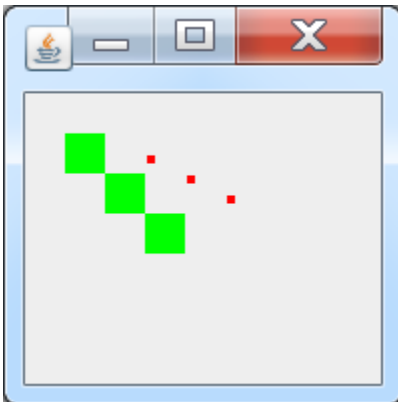
## Uruchomienie programu **Rys1\_3** – należy zwrócić uwagę na kolejność wyświetlanych figur

run:

```
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=20, y=20}, X=20, Y=20, odleglosc=28.284271247461902
false, ze jestem kwadratem, bo jestem Punkt{x=60, y=30}, X=60, Y=30, odleglosc=0.0
false, ze jestem kwadratem, bo jestem Punkt{x=80, y=40}, X=80, Y=40, odleglosc=0.0
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=40, y=40}, X=40, Y=40, odleglosc=56.568542494923804
false, ze jestem kwadratem, bo jestem Punkt{x=100, y=50}, X=100, Y=50, odleglosc=0.0
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=60, y=60}, X=60, Y=60, odleglosc=84.8528137423857
poszukanie punktu: true
poszukanie kwadratu: true
poszukanie punktu: false
zaznaczono false
zaznaczono true
zaznaczono true
przesuniето true
Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=35, y=35}
przesuniето true
Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=95, y=2}
przesuniето true
```



**Wynik działania metody `rysunek_konsola`.** Zastosowany pojemnik typu **TreeSet** wstawia elementy wg generowanej wartości przez metodę **compareTo** każdego elementu wstawianego do pojemnika podczas sortowania wg wartości składowych `x` i `y`. Obiekty były wstawiane na przemian: typu `Punkt` i typu `Kwadrat`





**Zadanie 4.** Zastosowanie klasy **TreeMap** jako pojemnika na obiekty. Podział programu na pakiety wspierające budowę dwuwarstwowej aplikacji – wykonanie programu **Rys1\_4**

1. Należy wykonać kopię projektu **Rys1\_1** jako **Rys1\_4** (podobnie jak na slajdzie 16).
2. Należy zmodyfikować nazwy klas: **Obraz1\_1** na **Obraz1\_4** (pakiet GUI), **FiguryHashSet** na **FiguryTreeMap** w pakiecie **grafika** wg slajdu 17
3. Należy zmodyfikować kod klas: **Figury**, **FiguryTreeMap**, **Figury\_panel** oraz **Obraz1\_4** dostosowując je do nowego typu pojemnika **TreeMap** podobnie jak w przypadku pojemnika **ArrayList** i uwzględniając typ interfejsu **Map** (slajdy 26).
4. **Należy dodać własną figurę: koło, trójkąt lub prostokąt zachowując zasady polimorfizmu i dodać metodę lezy\_na wg slajdu 8 w celu wprowadzenia hermetyzacji do metody Clicked.**

Pakiet **grafika**: w pliku **Figury.java** należy wkleić kod klasy podany poniżej, zaznaczony na czerwono, zależny od zmiany typu pojemnika

```
package grafika;
```

```
import figury.Kwadrat;
```

```
import figury.Punkt;
```

```
import java.awt.Graphics;
```

```
import java.util.Map;
```

```
public class Figury {
```

```
protected int N = 3;
```

```
public Map<Punkt,Punkt> figury;
```

```
protected Punkt biezacy;
```

```
public Punkt getBiezacy() { return biezacy; }
```

```
public void polozenie() {
```

```
for (Punkt figura : figury.values()) {
```

```
boolean p = figura instanceof Kwadrat;
```

```
System.out.println(
```

```
    p + ", ze jestem kwadratem, bo jestem " + figura.toString()
```

```
    + ", X=" + figura.getX()
```

```
    + ", Y=" + figura.getY()
```

```
    + ", odleglosc=" + figura.odleglosc(figura));
```

```
    }
```

```
}
```

Nowy typ interfejsu pojemnika  
TreeMap - **Map**

Pobieranie posortowanych danych z pojemnika  
na dane za pomocą zaimplementowanej metody  
**compareTo** w klasie **Punkt**

```

public boolean Clicked(int x_, int y_)
{
    for (Punkt figura : figury.values()) {
        if (figura.getX() + figura.getDI() >= x_
            && figura.getX() <= x_
            && figura.getY() + figura.getDI() >= y_
            && figura.getY() <= y_) {
            biezacy = figura;
            return true; }
        }
    return false;
}

```

```

public void rysuj_figury(Graphics g) {
    for (Punkt figura : figury.values()) {
        figura.rysuj(g);
    }
}

```

```

public boolean przesun(int x, int y, int dl, int szer) {
    if (biezacy != null) {
        biezacy.przesun(x, y, dl, szer);
        return true; }
    return false;
}
}

```

Pakiet **grafika**: w pliku **FiguryTreeMap.java** należy wkleić kod klasy podany poniżej, zaznaczony na czerwono, zależny od zmiany typu pojemnika

```
package grafika;
```

```
import figury.Kwadrat;
```

```
import figury.Punkt;
```

```
import java.util.TreeMap;
```

```
public class FiguryTreeMap extends Figury {
```

```
    public void pojemnik() {
```

```
        figury = new TreeMap();
```

```
    }
```

```
    public boolean wyszukaj(Punkt p) {
```

```
        return figury.containsKey(p); }
```

```
    public boolean wyszukaj(int dane[]) {
```

```
        Punkt p;
```

```
        if (dane[0] == 0) {
```

```
            p = new Punkt(dane[1], dane[2]);
```

```
        } else {
```

```
            p = new Kwadrat(dane[1], dane[2], dane[3]);
```

```
        }
```

```
        return wyszukaj(p);
```

```
    }
```

Pobieranie wyszukanych danych z pojemnika na **klucze** danych

```

public void wypelnij() {
    for (int i = 0; i < N; i++) {
        figury.put(new Punkt(20 * (N + i), 10 * (N + i)), new Punkt(20 * (N + i), 10 * (N + i)));
        figury.put(new Kwadrat((i + 1) * 20, (i + 1) * 20, 20), new Kwadrat((i + 1) * 20, (i + 1) * 20, 20));
    }
}

public void init() {
    pojemnik();
    wypelnij();
    polozenie();
}
}

```

Wstawianie danych do pojemnika typu **TreeMap** - każda dana składa się z **klucza (typu Punkt lub Kwadrat) i danej (typu Punkt lub Kwadrat) o tych samych wartościach składowych**, a dane będą posortowane wg **klucza** – w zadaniu wg składowych x i y.

## Uruchomienie programu Rys1\_4

run:

true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=20, y=20}, X=20, Y=20, odleglosc=28.284271247461902  
false, ze jestem kwadratem, bo jestem Punkt{x=60, y=30}, X=60, Y=30, odleglosc=0.0  
false, ze jestem kwadratem, bo jestem Punkt{x=80, y=40}, X=80, Y=40, odleglosc=0.0  
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=40, y=40}, X=40, Y=40, odleglosc=56.568542494923804  
false, ze jestem kwadratem, bo jestem Punkt{x=100, y=50}, X=100, Y=50, odleglosc=0.0  
true, ze jestem kwadratem, bo jestem Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=60, y=60}, X=60, Y=60, odleglosc=84.8528137423857  
poszukanie punktu: true  
poszukanie kwadratu: true  
poszukanie punktu: false  
zaznaczono false  
zaznaczono true  
zaznaczono true  
przesuniето true  
Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=35, y=35}  
przesuniето true  
Kwadrat{dlugosc=20} i dziedzicze od Punkt{x=95, y=2}  
przesuniето true

← **Wynik działania metody `rysunek_konsola`.** Zastosowany pojemnik typu **TreeSet** wstawia elementy wg generowanej wartości przez metodę **`compareTo`** klucza każdego elementu wstawianego do pojemnika podczas sortowania wg wartości składowych `x` i `y`. Obiekty były wstawiane na przemian: typu `Punkt` i typu `Kwadrat`. **Klucz i dane każdego elementu pojemnika są takiego samego typu i posiadają te same wartości składowych `x` i `y`.**

