DOSTĘP DO METOD I ZMIENNYCH

Dostęp do zmiennych (na podstawie:L.Lemay,R.Cadenhead,Java 2 dla każdego, Helion 2001)

Zakres „widzialności” metody lub zmiennej	public	protected	domyślny	private
Klasa	tak	tak	tak	tak
Pakiet	tak	tak	tak	nie
Inny pakiet	tak	nie	nie	nie
Podklasa, pakiet	tak	tak	tak	nie
Podklasa, inny pakiet	tak	tak	nie	nie

DZIEDZICZENIE
PRZECIĄŻANIE METOD, PRZEDEFINIOWANIE METOD

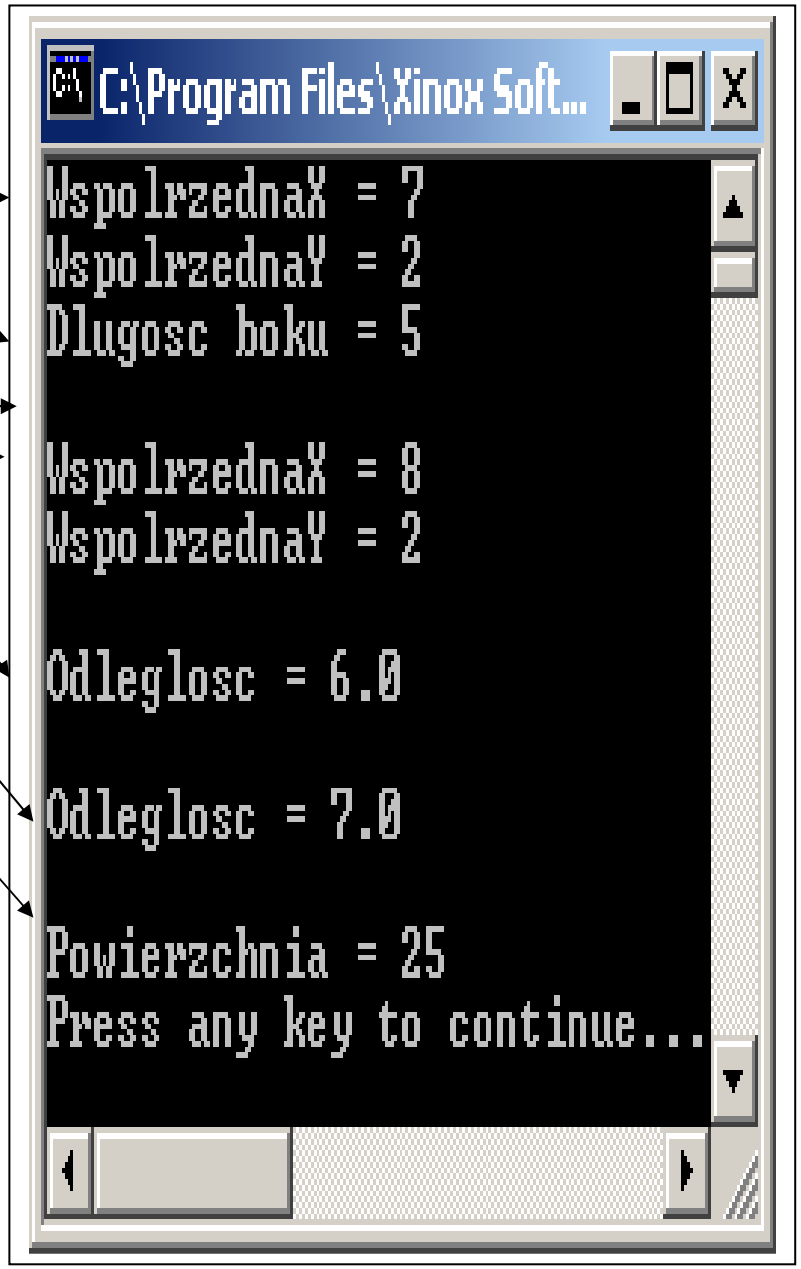
1) Dziedziczenie
Przykład 1 – dziedziczenie metod w klasie pochodnej

```
class Punkt
{  protected int x, y;

  public Punkt(int wspX, int wspY)
      {   x = wspX;       y = wspY;}

  public void zmien(int wspX, int wspY)
      {   x = wspX;       y = wspY;}

 public int podajX()
      { return x;  }

public  int podajY()
      { return y; }

 public void przesun(int dx, int dy)
      {   x+=dx;   y+=dy;  }

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

```
class Kwadrat extends Punkt          //klasa Kwadrat, zwana klasą pochodną, dziedziczy od klasy Punkt, zwanej klasą bazową
 {
    protected int dlugosc;

  Kwadrat(int wspX, int wspY, int dlugosc_)
   {
     super(wspX,wspY);                          //wywołanie konstruktora klasy bazowej
    dlugosc=dlugosc_;
   }          //ponieważ nie ma konstruktora domyślnego (jawnego lub niejawnego konstruktora bez parametrów) w klasie
                                    // bazowej Punkt-konstruktora domyślnego się nie wywołuje
  int podajDl()
    {
      return dlugosc;
     }

  int pole()
    {
      return dlugosc*dlugosc;
     }
}
```

```
public class punkt_
{
  public static void main (String[] args)
  {
    Kwadrat k1 = new Kwadrat(7,2,5);
    System.out.println("WspolrzednaX = "+ k1.podajX());
    System.out.println("WspolrzednaY = "+ k1.podajY());
     System.out.println("Dlugosc boku = "+ k1.podajDl());
     Punkt p2 = new Punkt(8,2);
     System.out.println("\nWspolrzednaX = "+ p2.podajX());
     System.out.println("WspolrzednaY = "+ p2.podajY());
     p2.zmien(1,2);
     System.out.println("\nOdleglosc = "+ p2.odleglosc(k1));
     k1.przesun(1,0);
     System.out.println("\nOdleglosc = "+ k1.odleglosc(p2));
     System.out.println("\nPowierzchnia = "+ k1.pole());
 }
}
```

Te metody są wywołane dzięki dziedziczeniu składowych od klasy Punkt



Obiekt klasy *Kwadrat* posiada metody klasy Punkt i swoje własne (podajDl(), pole() oraz konstruktor Kwadrat(**int, int, int**). W programie w konstruktorze klasy Kwadrat polecenie **super** pozwala wywołać konstrutor dziedziczonej klasy *Punkt*. Konstruktory domyślne nie trzeba wywoływać (są wywoływane niejawnie), pozostałe konstruktory, jeśli nie ma domyślnych, trzeba zawsze wywołać za pomocą słowa **super**, jako pierwsze wywołanie w konstruktorze klasy pochodnej.

Wywołanie p2.odleglosc(k1) zawiera parametr aktualny metody jako referencję do obiektu klasy pochodnej Kwadrat, która może być podstawiona za parametr typu klasy bazowej Punkt.

2) Przedefiniowanie i przeciążenie metod

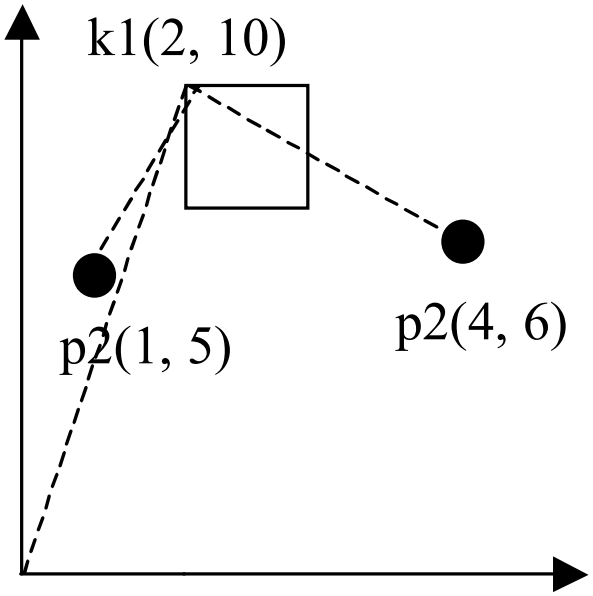
Przedefiniowanie nazwy metody **met1(...)** dziedziczonej wymaga :

- identycznego nagłówka **met1(...)** w metodzie klasy dziedziczącej
- jest to domyślna metoda klasy dziedziczącej
- metodę bazową w ciele metod klasy dziedziczącej wywołuje się: **super**.met1(...)

Przeciążenie nazwy metody **met2(...)** wymaga

- identycznej nazwy **met2**
- różnej liczby parametrów
- różnych typów parametrów, jeśli ich liczba jest identyczna
- typ wyniku zwracanego przez **return** nie przeciąża nazwy metody

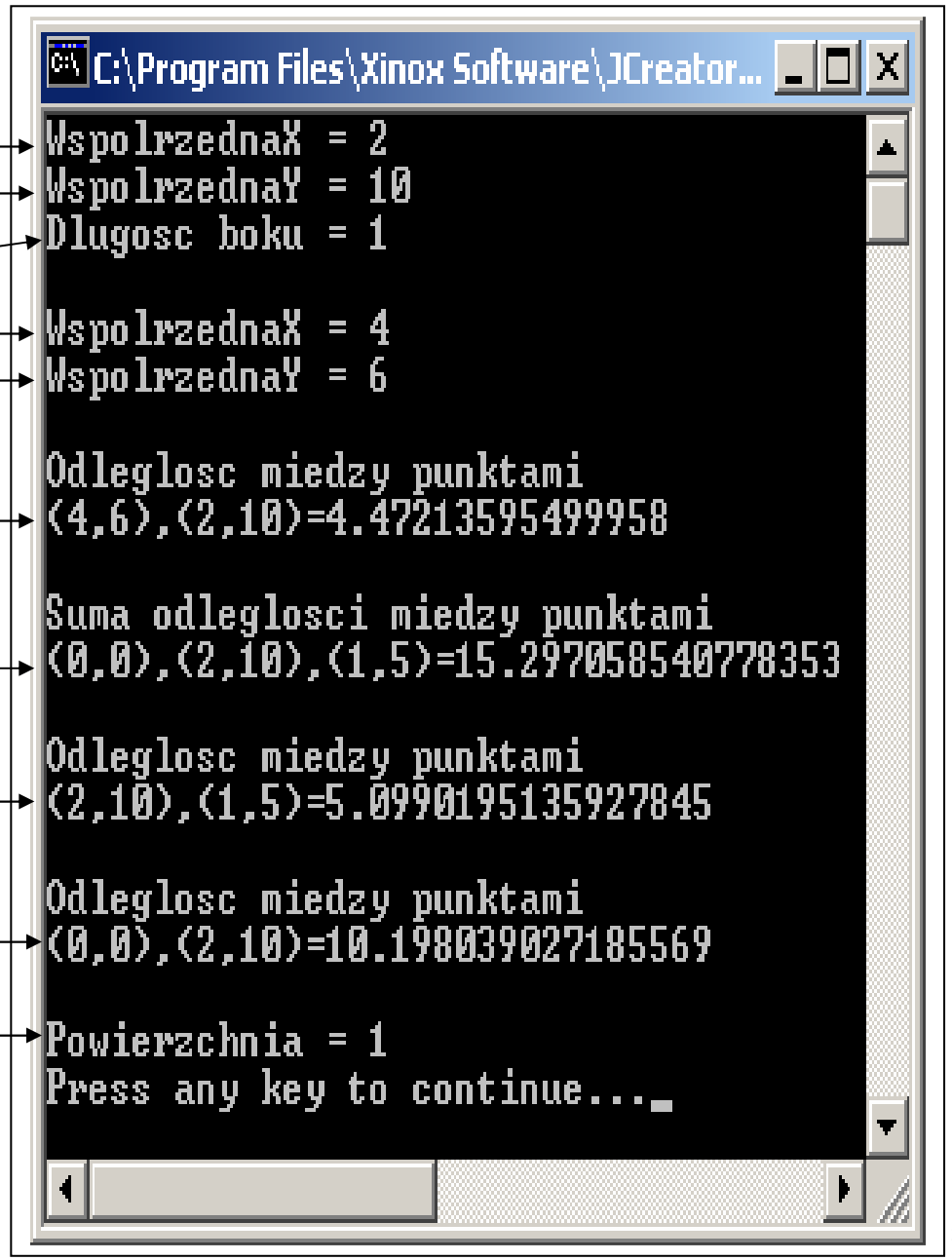
Przykład 2 - wykonanie programu prezentującego dziedziczenie, przedefiniowanie i przeciążenie metod




```
import java.lang.*;
class Punkt
{ protected int x, y;

  public Punkt(int wspX, int wspY)
  {  x = wspX; y = wspY;}

 public void zmien(int wspX, int wspY)
  {  x = wspX; y = wspY;}

public  int podajX()
  { return x;  }

 public int podajY()
  { return y;  }

 public void przesun(int dx, int dy)
  {  x+= dx;
     y+= dy;  }
public int pole ()
  { return 0;}

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

}
```

```
class Kwadrat extends Punkt
 {
  protected int dlugosc;

  public Kwadrat(int wspX, int wspY, int dlugosc_)
     { super(wspX, wspY);`                    //wywołanie konstruktora od dziedziczonej klasy Punkt
       dlugosc = dlugosc_;  }

  public int podajDl()
    {  return dlugosc; }

  public int pole()
    { return dlugosc*dlugosc;  }

  public double odleglosc()                   //przeciążenie nazwy metody dziedziczonej
    { return Math.sqrt(x*x+y*y);   }

  public double odleglosc(Punkt p)            //przedefiniowanie nazwy metody dziedziczonej
    { return odleglosc()                      //wywołanie metody przeciążonej
             + super.odleglosc(p);  }         //wywołanie metody dziedziczonej
                                              // gdy jest przedefiniowana

  public double odleglosc(Punkt p, int i)     //przeciążenie nazwy metody dziedziczonej
    { return super.odleglosc(p); }            //wywołanie metody dziedziczonej,
                                              // gdy jest przedefiniowana
 }
```

```
public class punkt1                          //klasa publiczna, nieabstrakcyjna, niefinalna
{
 public static void main (String[] args)
 {
  Kwadrat k1 = new Kwadrat(2,10,1);
  System.out.println("WspolrzednaX = "+ k1.podajX());
  System.out.println("WspolrzednaY = "+ k1.podajY());
  System.out.println("Dlugosc boku = " + k1.podajDl());
  Punkt p2 = new Punkt(4,6);
  System.out.println("\nWspolrzednaX = "+ p2.podajX());
  System.out.println("WspolrzednaY = " + p2.podajY());
  System.out.println(
     "\nOdleglosc miedzy punktami\n(4,6),(2,10)="
     + p2.odleglosc(k1));
  p2.zmien(1,5);
  System.out.println(
   "\nSuma odleglosci miedzy punktami\n(0,0),(2,10),(1,5)="
     +k1.odleglosc(p2));
  System.out.println(
     "\nOdleglosc miedzy punktami\n(2,10),(1,5)="
     + k1.odleglosc(p2,0));
  System.out.println(
     "\nOdleglosc miedzy punktami\n(0,0),(2,10)="
     + k1.odleglosc());

  System.out.println("\nPowierzchnia = "+ k1.pole());
 }
```



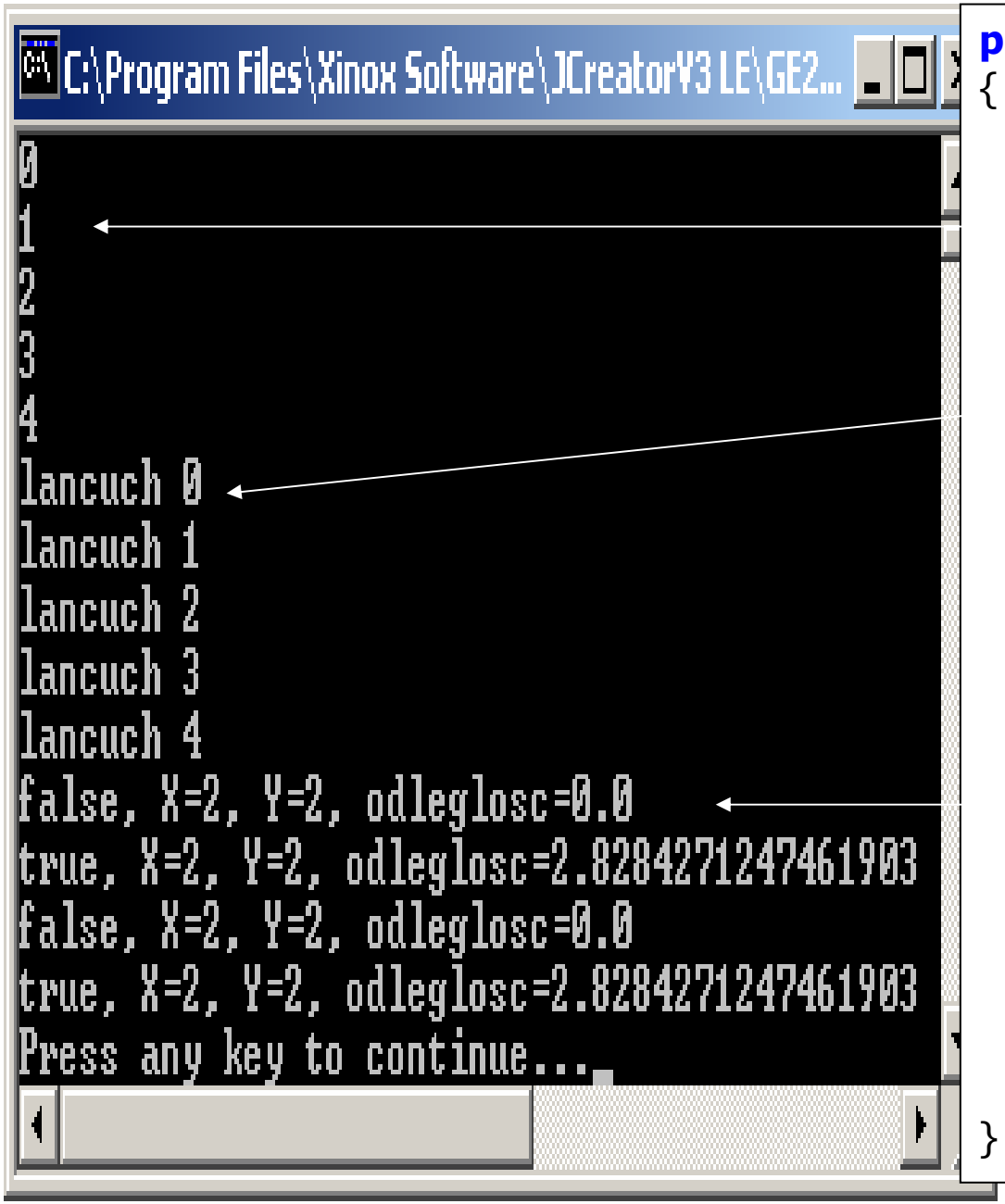
TABLICE (2), POLIMORFIZM

Przykład 3 - przechowanie w tablicy elementów obiektowych – polimorfizm (przesłanianie metod)

```
import java.lang.*;
class Punkt
{ protected int x, y;
  public Punkt(int wspX, int wspY)
       { x=wspX; y = wspY;}
  public int podajX()
     { return x;}
  public int podajY()
     { return y;}
  public double odleglosc(Punkt p)
   {  return Math.sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y)); }
}
  class Kwadrat extends Punkt
   { protected int dlugosc;
      public Kwadrat(int wspX, int wspY, int dlugosc_)
        { super(wspX,wspY);
          dlugosc=dlugosc_;}
     public  double odleglosc()
        { return Math.sqrt(x*x+y*y); }
     public double odleglosc(Punkt p)
        { return odleglosc()+ super.odleglosc(p);}
     public int podajDl()  { return dlugosc;}  }
```

Operacje wynikające z polimorfizmu:

- Można przypisać do referencji klasy podstawowej referencje obiektów klas pochodnych
- Referencja **figury[i]** może zawierać referencję do obiektu typu **Punkt** lub **Kwadrat** – interpreter rozróżnia te referencje i wywołuje metodę z właściwej klasy. Jest to możliwe dzięki przesłanianiu metod **odleglosc.** Kompilator sprawdza, czy dana metoda jest w klasie definiującej elementy tablicy.




```
public class Tablice
{ public static void main(String args[])
  {final int N=5;
    int liczby [] = new int [N];
      for (int i=0; i<liczby.length; i++)
              {  liczby[i]= i;
                 System.out.println(liczby[i]); }
   String nazwy[]=new String[N];
       for (int i=0; i<nazwy.length; i++)
              {  nazwy[i]=new String("lancuch "+i);
                 System.out.println(nazwy[i]);}
   Punkt figury[]=new Punkt[N];
    for (int i=0; i<figury.length; i++)
     { figury[i]=new Punkt(2,2);
       if (i<figury.length-1)
          figury[++i]= new Kwadrat (2,2,2); }
    for (int i=0; i<figury.length-1;i++)
     {boolean p=figury[i] instanceof Kwadrat;
      System.out.println(p+
       ",X="+figury[i].podajX()+
       ", Y="+figury[i].podajY()+
       ", odleglosc="+figury[i].odleglosc(figury[i+1]));
      }
}
```

Przykład 4 - przechowanie w tablicy elementów obiektowych – polimorfizm (przesłanianie metod)

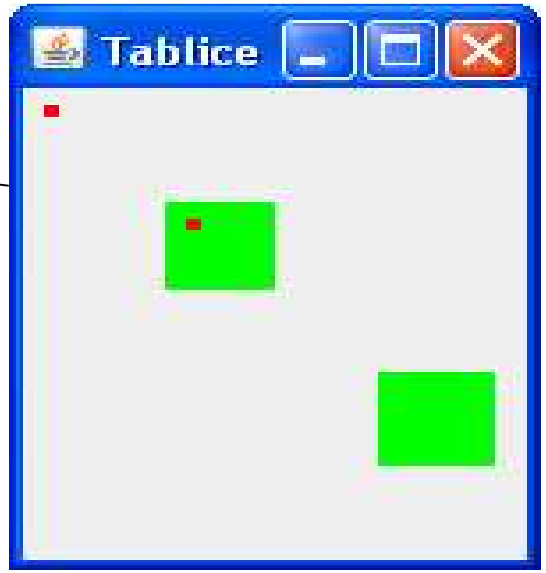
```
import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Punkt
{ protected int x, y;
  public Punkt(int wspX, int wspY)
     {x = wspX; y = wspY;}
  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); }
 }
class Kwadrat extends Punkt
{   protected int dlugosc;
   public Kwadrat(int wspX, int wspY, int dl)
      { super(wspX,wspY);
        dlugosc=dl; }
   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); }
}
```

```
class Figury
{ protected int N=4;
  protected Punkt figury[]=new Punkt[N];
  public Figury()
   { for (int i=0; i<figury.length; i++)
     { figury[i]= new Punkt(i*20+5, i*20+5); //1
       if (i<figury.length-1)
        figury[++i]=
        new Kwadrat (i*30+10, i*30+10, i+30); //2
     } }
    public void rysuj(Graphics g)
      {  for (int i=0; i<figury.length; i++)
              figury[i].rysuj(g); }
  }
```

Operacje wynikające z polimorfizmu:

- Można przypisać do referencji klasy podstawowej referencje obiektów klas pochodnych (//1, //2)
- Referencja **figury[i]** może zawierać referencję do obiektu typu **Punkt** lub **Kwadrat** – rozróżnia się te referencje i wywołuje metodę **rysuj** z właściwej klasy dzięki przesłanianiu metod **rysuj**.

```
class Panel extends JPanel
 { Figury rys= new Figury();
   public void  paintComponent(Graphics g)
     {
        super.paintComponent(g);
        rys.rysuj(g);
     }
 }

public class Tablice extends JFrame
 {  public Tablice()
      { super ("Tablice");
        setSize(150,200);
        setDefaultCloseOperation
          (JFrame.EXIT_ON_CLOSE);
        Panel obraz=new Panel();
        setContentPane(obraz);
 }
  public static void main(String args[])
 { Tablice tab=new Tablice();
   tab.setVisible(true);
 }
}
```



Obiekt klasy **JFrame** używa obiektu typu **JPanel**. Klasa **Tablice** dziedzicząca po klasie **JFrame** używa obiekt typu **Panel** dziedziczący po **JPanel**. Klasa **Panel** przesłoniła metodę paintComponent.
Używa ona wyspecjalizowanego obiektu typu **Figury** do rysowania figur na swojej powierzchni.
Każda klasa dziedzicząca po **JFrame** używa metody painComponent wywoływanej od obiektu klasy z rodziny **JPanel** bez potrzeby rozróżniania typu tych obiektów, do rysowania treści graficznej ramki.