Programowanie obiektowe

Wykład 13:
RTTI

Po co informacja o typie obiektu?

- Polimorfizm
- Kolekcje
- Deserializacja

Przykład: rzutowanie w dół

```
import java.util.*;

class Kształt {
  void rysuj() {
    System.out.println(this + ".rysuj()");
  }
}

class Koło extends Kształt {
  public String toString() { return "Koło"; }
}

class Kwadrat extends Kształt {
  public String toString() { return "Kwadrat"; }
}

class Trójkąt extends Kształt {
  public String toString() { return "Trójkąt"; }
}
```

```
public class Kształty {
  public static void main(String[ ] args) {
    List s = new ArrayList();
    s.add(new Koło());
    s.add(new Kwadrat());
    s.add(new Trójkąt());
    Iterator e = s.iterator();
    while(e.hasNext())
      ((Kształt)e.next()).rysuj();
  }
}
```

A co gdybyśmy chcieli rzutować np. na Koło?

Wyjątek: ClassCastException !

Jak sprawdzić „typ" obiektu?

Operator *instanceof*

```
while(e.hasNext()) {
  Object o=e.next();
  if (o instanceof Koło) ((Kolo)o).draw();
  if (o instanceof Kwadrat) ((Kwadrat)o).draw();
  if (o instanceof Trójkąt) ((Trójkąt)o).draw();
}
```

Klasa Class

- Zawiera metody udostępniające wszystkie informacje o klasie
- Instancja klasy tworzona w chwili pierwszego odwołania do klasy
- Przechowywana w pliku .class
- Używana do tworzenia wszystkich obiektów danej klasy

Metoda forName()

Utworzenie instancji klasy Class na podstawie nazwy klasy:

```
Class c=Class.forName(”Koło”);
```

Wykonuje inicjalizację pól statycznych!

Przykład: sprawdzenie czy klasa jest dostępna

```
try {
  Class.forName(”Klasa");
} catch(ClassNotFoundException e) {
   System.out.println(”Nie można znaleźć pliku Klasa.class");
}
```

Uzyskanie dostępu do obiektu Class

Class c=o.getClass();
Class c=Kolo.class

String name=c.getName();
Object o=c.newInstance();
c.isInstance(o) → Koło.class.isInstance(o) == o instanceof Koło

Class dla typów prostych

boolean.class	**==**	**Boolean.TYPE**
char.class	**==**	**Character.TYPE**
byte.class	**==**	**Byte.TYPE**
short.class	**==**	**Short.TYPE**
int.class	**==**	**Integer.TYPE**
long.class	**==**	**Long.TYPE**
float.class	**==**	**Float.TYPE**
double.class	**==**	**Double.TYPE**
void.class	**==**	**Void.TYPE**

Przykłady

- Class c = ”text".getClass();
  - Class dla klasy String
- byte[ ] bytes = new byte[1024];
  Class c = bytes.getClass();
- Class c = int[][][].class;
  - Class dla tablicy
- Set<String> s = new HashSet<String>();
  Class c = s.getClass();
  - Class dla klasy HashSet

Odzwierciedlenia

Pakiet: java.lang.reflect :

public final class **Constructor**
public final class **Field**
get(), set()
public final class **Method**
invoke()

```
Class c=o.getClass();
        Class sup=c.getSuperclass();
        Class [ ] i=c.getInterfaces();
        Method [ ] m=c.getMethods();        -- również dziedziczone!!!
        Field [ ] f=c.getFields();
        Constructor[ ] constr=c.getConstructors();
```

Przykład

```
import java.lang.reflect.*;
import java.util.regex.*;

public class ShowMethods {
  private static final String usage =
    "usage: \n" +
    "ShowMethods qualified.class.name\n" +
    "To show all methods in class or: \n" +
    "ShowMethods qualified.class.name word\n" +
    "To search for methods involving 'word'";
  private static Pattern p = Pattern.compile("\\w+\\.");
  public static void main(String[ ] args) {
    if(args.length < 1) {
      System.out.println(usage);
      System.exit(0);
    }
    int lines = 0;
    try {
      Class c = Class.forName(args[0]);
      Method[] m = c.getMethods();
      Constructor[] ctor = c.getConstructors();
```

```
      if(args.length == 1) {
        for(int i = 0; i < m.length; i++)
          System.out.println(
            p.matcher(m[i].toString()).replaceAll(""));
        for(int i = 0; i < ctor.length; i++)
          System.out.println(
            p.matcher(ctor[i].toString()).replaceAll(""));
        lines = m.length + ctor.length;
      } else {
        for(int i = 0; i < m.length; i++)
          if(m[i].toString().indexOf(args[1]) != -1) {
            System.out.println(
              p.matcher(m[i].toString()).replaceAll(""));
            lines++;
          }
        for(int i = 0; i < ctor.length; i++)
          if(ctor[i].toString().indexOf(args[1]) != -1) {
            System.out.println(p.matcher(
              ctor[i].toString()).replaceAll(""));
            lines++;
          }
      }
    } catch(ClassNotFoundException e) {
      System.out.println("No such class: " + e);
    }
  }
}
```


Pytania

1. Na czym polega identyfikacja typu podczas wykonania?
2. Kiedy potrzebna jest identyfikacja typu?

Koniec