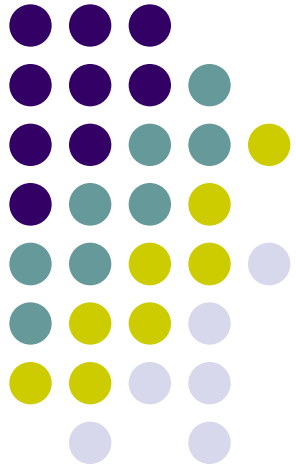
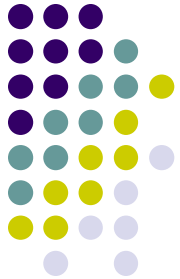
Мова програмування Java та технології J2EE
Модуль “Мова програмування Java”



Лекція 4.

Узагальнене програмування на мові Java (Generics).

Приклад без застосування узагальнень



- Перевантаження коду змінними типу Object

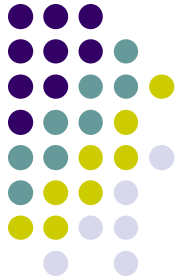
```
public class Box {
    private Object object;
    public void add(Object object) { this.object = object; }
    public Object get() { return object; }
}
```

- Перевантаження коду приведенням типів

```
public static void main(String[] args) {
    Box integerBox = new Box(); // домовимося передавати в
                                // Box значення Integer
    integerBox.add("10"); // увага – це значення типу String
    ...
    Integer someInteger = (Integer)integerBox.get();// помилка
                                                    //часу виконання
}
```

- Якщо негаразд із типами - **помилка часу виконання**
  - `java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer`

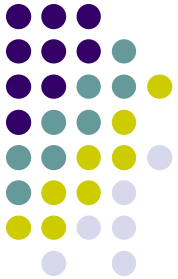
Узагальнення (Generics)



- Основні класи задач, які потребують застосування узагальнень:
  - Розробка функцій-утиліт для колекцій (пошук, max, min, avg, sum тощо)
  - Розробка контейнерів для об’єктів різних типів (стек, колекція тощо)

- Мотивація для generics:
  - Отримувати на етапі компіляції помилки приведення типів
  - Обійтись без приведення типів “вручну”
  - Отримати більш безпечний код, який краще пишеться та читається

Історія появи узагальнень в Java



- Узагальнення побачили світ в J2SE 5 (2004р.)

- Задача - розширити систему типів мови, що широко застосовується і до якої висуваються вимоги жорсткої зворотної сумісності
- Роботу розпочато у 1999р.
- Деякі деталі із проробки задачі:
  - Специфікація “JSR-014: Adding Generics to the Java Programming Language” розроблялася протягом 1999-2004
  - Розширення системи типів підстановочними типами (wildcards) здійснено у співпраці Sun та університету м.Орхус (Данія)
    - Цікаво – один із відомих уродженців м.Орхус – Бйорн Страуструп, автор мови C++

Приклад застосування узагальнень

- Замість `Object` застосовуємо “типи-параметри”

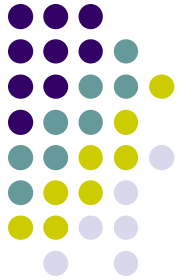
```
public class Box<T> {
    private T t;
    public void add(T t) { this.t = t; }
    public T get() { return t; }
}
```

- Не потрібно приводити типи

```
public static void main(String[] args) {
    Box<Integer> integerBox = new Box<Integer>();
    integerBox.add("10");                    // Помилка компіляції
    Integer someInteger = integerBox.get(); // Не потрібне приведення
                                             // типів
}
```

- Негаразд із типами - помилка компіляції

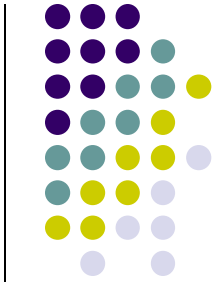
Реалізація узагальнень в Java



- Це елементи мови
- Це функціональність **компілятора**, яка дозволяє виявити певні помилки на стадії компіляції

- Це **не** функціональність **JVM**
- **Type erasure** - На стадії виконання (runtime) уся інформація про узагальнення стирається
  - Через вимоги жорсткої зворотної сумісності – старий байт-код повинен працювати на нових JVM
- Узагальнення не потребують додаткових ресурсів часу виконання

Елементи мови, які узагальнюються



- Що може бути узагальнене
  - Класи
    - але не всі, див. нижче
  - Інтерфейси
  - Методи
  - Конструктори
- Які типи можуть бути параметрами для узагальнення
  - Типи-посилання (класи, інтерфейси, масиви)
- Які типи не можуть бути параметрами для узагальнення
  - Примітивні типи (але класи-оболонки можуть)
- Які класи не можуть бути узагальнені
  - Enum
    - Чому?
  - Клас Throwable та його нащадки
    - Обмеження викликане тим, що механізм catch у JVM не працює з параметризованими класами

Узагальнені типи

- **Узагальнені типи – узагальнені класи та узагальнені інтерфейси**

```
public class Box<T> {
    private T t;
    public void add(T t) { this.t = t; }
    public T get() { return t; }
}
```

`Box` – узагальнений клас, який вводить **змінну типу** `T`

- Декілька змінних типу

```
class Suitecase<T,U> {…}
```

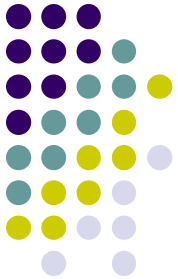
- **Успадкування** для узагальнених класів/інтерфейсів

```
Suitecase<T,U> extends Box<T> {…}
```

- Угода щодо назв змінних типів

```
E – Element
(використовується у Java
Collections Framework)
K – Key
T – Type
V - Value
```

Узагальнені типи. Продовження

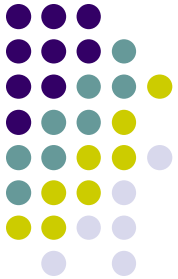


Box<T> - узагальнений клас із змінною типу T
Box<Integer> - параметризований тип,
із параметром (аргументом) Integer

- Приклади
  Vector<String>
  Seq<Seq<A>>
  Collection<Integer>
  Pair<String,String>
  Iterator<**int[]**> - **параметризація масивом**

- Виклик конструктора
  Box<Integer> integerBox = new Box<Integer>();

Узагальнені методи



- Узагальнений метод

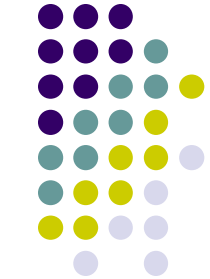
```
class Inspector {

    public <T> void inspect(T t) {
        System.out.println(t.getClass().getName());
    }
}
```

- Виклик методу

```
Inspector i = new Inspector();
String s = "Hello";
i.inspect(s);               // короткий синтаксис
i.<String>inspect(s);       // повний синтаксис
```

- Узагальнення конструкторів відбувається аналогічно

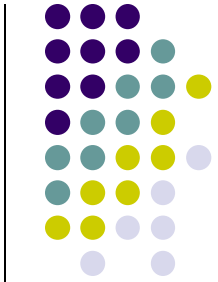
Обмеження для змінних типу



- Подібного немає в C++
- **`extends &`**
  - **`extends`** – означає, що параметр типу повинен успадковувати вказаний клас чи реалізовувати вказані інтерфейси
  - **`&`** - дозволяє вказати декілька типів, які мають бути успадковані або реалізовані (один клас, декілька інтерфейсів). “,” застосувати не можна, оскільки це роздільник між змінними типу

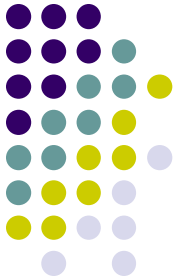
```
class Inspector {
    public <T extends Number&Comparable> void inspect(T t) {…}
}

Inspector i = new Inspector();
String s = "Hello";
i.inspect(s);           // помилка компіляції,
                        // оскільки s - це не Number & Comparable
```

Whildcards (підстановочні типи). Мотивація



```
public void boxTest(Box<Number> n) { ...}

boxTest(new Box<Integer>()); // compile error
boxTest(new Box<Double>());  // compile error
```

Whildcards (підстановочні типи). Мотивація



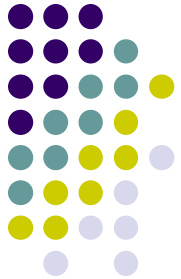
- Опція 1

```
public void boxTest(Box<?> n) { ... }

boxTest(new Box<Integer>()); // ok
boxTest(new Box<Double>());  // ok
boxTest(new Box<String>());  // ok?
```

- Опція 2

```
public void boxTest(Box<? extends Number> n) { ...}

boxTest(new Box<Integer>()); // ok
boxTest(new Box<Double>());  // ok
```

Wildcards details



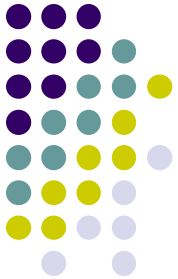
- Застосування “?”
  - тільки для тих type arguments
  - в полях/локальних змінних/методах/конструкторах/декларації класів

```
List<Integer> l = Arrays.asList(1, 2);
List<? extends Number> lNum = l;
```

Bounded wildcards

- **Bounded** wildcards
  - `<? extends` *Тип*`>` - будь-який тип-нащадок *Тип*
  - `<? super` *Тип*`>` - будь-який тип-пращур *Тип*

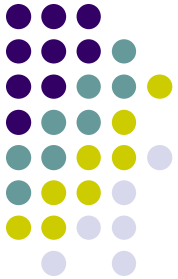
Let’s look to java source



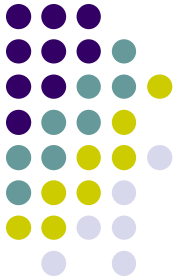
```
public class LinkedList<E> {
    public boolean addAll(Collection<? extends E> c) {…}
}
```

Deep dive to java source

```
public interface List<E> extends Collection<E>
    default void sort(Comparator<? super E> c) { … }
}
```



Див. наступний слайд для прикладу навіщо все це



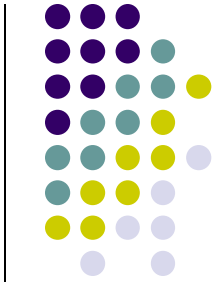
Це продовження попереднього слайду
Дано

```
class A {}

class AA extends A {}

class AAA extends AA {}

class AComparator implements Comparator<A> {
    public int compare(A o1, A o2) {  return 0; }
}

class AAComparator implements Comparator<AA> {
    public int compare(AA o1, AA o2) { return 0; }
}

class AAAComparator implements Comparator<AAA> {
    public int compare(AAA o1, AAA o2) { return 0;  }
}
```

Який рядок не буде компілюватися і чому

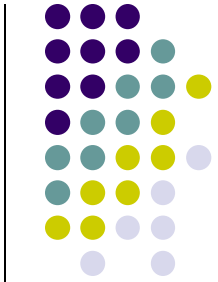
```
public static void main(String[] args) {

    List<AA> l = new ArrayList<>();
    l.sort(new AComparator());
    l.sort(new AAComparator());
    l.sort(new AAAComparator());
}
```

Real case of java generic constructor



```
class State <V> {
    V value;
}
interface TransitionFunction<V, S> {
    V transit (V calue, S signal);
}

class StateLauncher<V> {
    State <V> internalState;
    <S> StateLauncher (V value, TransitionFunction<V,S> transFunc, S signal) {
        internalState.value = transFunc.transit(value, signal);
    }
}

 public static void main(String[] args) {

     StateLauncher<Integer> sl = new StateLauncher<Integer>(
             1,
             new TransitionFunction<Integer, String>() {
                 public Integer transit(Integer value, String signal) {
                     return value + signal.hashCode();
                 }
             },
             "sample signal");
 }
```

Література



- **The Java Tutorial.** http://download.oracle.com/javase/tutorial/java/TOC.html
- James Gosling, Bill Joy, Guy Steele. **The Java Language Specification**. - Addison Wesley. - 3 edition. - 2005. - 688p. -http://java.sun.com/docs/books/jls/