СТВОРЕННЯ ТА ВИКОРИСТАННЯ АНОТАЦІЙ У КОДІ

Під анотуванням розуміють прив’язку метаданих (даних, що описують інші дані).
Наприклад, можна ідентифікувати методи, які не повністю реалізовані, щоб не забути завершити реалізацію.

Огляд анотацій

- Анотація – екземпляр анотованого типу, що асоціює метадані з елементом додатку.
  - В коді позначається за допомогою префіксу @.
  - Наприклад, @Readonly – анотація, а Readonly – її тип.
- **Зауважте** Можна використовувати анотації для асоціації метаданих з конструкторами, полями, локальними змінними, методами, пакетами, параметрами та типами (annotation, class, enum, interface).
- Компілятор підтримує типи анотацій Override, Deprecated та SuppressWarnings.
  - Містяться в пакеті java.lang.
- Анотації @Override корисні для вирішення, чи буде метод підкласу переозначувати, а не перевантажувати метод суперкласу.

```
@Override
public void draw(int color)
{
   // drawing code
}
```

Огляд анотацій

```
/**
 * Allocates a <code>Date</code> object and initializes it so that
 * it represents midnight, local time, at the beginning of the day
 * specified by the <code>year</code>, <code>month</code>, and
 * <code>date</code> arguments.
 *
 * @param   year    the year minus 1900.
 * @param   month   the month between 0-11.
 * @param   date    the day of the month between 1-31.
 * @see     java.util.Calendar
 * @deprecated As of JDK version 1.1,
 * replaced by <code>Calendar.set(year + 1900, month, date)</code>
 * or <code>GregorianCalendar(year + 1900, month, date)</code>.
 */
@Deprecated
public Date(int year, int month, int date)
{
    this(year, month, date, 0, 0, 0);
}
```

- Анотації @Deprecated корисні для позначення застарілості елементу та небажаності його подальшого використання.
  - Компілятор попереджає, коли до застарілого коду намагається отримати доступ актуальний (nondeprecated) код.
- На відміну від нього javadoc-тег @deprecated та пов’язаний з ним тег попереджає проти використання застарілого елементу і каже, що потрібно використовувати взамін.

Посилання на Deprecated поле з оголошення того ж класу

```
public class Employee
{
   /**
    * Employee's name
    * @deprecated New version uses firstName and lastName fields.
    */
   @Deprecated
   String name;
   String firstName;
   String lastName;

   public static void main(String[] args)
   {
      Employee emp = new Employee();
      emp.name = "John Doe";
   }
}
```

- Компілятор пригнічує повідомлення, коли compilation unit (зазвичай клас або інтерфейс) звертається до застарілого класу, методу чи поля.
  - Ця можливість дозволяє змінювати legacy APIs без утворення deprecation warnings.
- У лістингу оголошено клас Employee з полем name, яке застаріло.
  - Хоч метод main() звертається до name, компілятор пригнічує deprecation warning, оскільки deprecation та reference виникають в одному класі.

Посилання на застаріле поле з оголошення іншого класу

```
class Employee
{
   /**
    * Employee's name
    * @deprecated New version uses firstName and lastName
    */
   @Deprecated
   String name;
   String firstName;
   String lastName;
}

public class UseEmployee
{
   public static void main(String[] args)
   {
      Employee emp = new Employee();
      emp.name = "John Doe";
   }
}
```

- Припустимо, після рефакторингу з'явився новий клас UseEmployee, а метод main() перемістився в нього.
  - Після компіляції з'являться повідомлення

```
Employee.java:18: warning: [deprecation] name in Employee has been deprecated
      emp.name = "John Doe";
         ^
1 warning
```

```
Note: UseEmployee.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

При компіляції в командному рядку доведеться додавати ключ -Xlint:deprecation:
javac -Xlint:deprecation UseEmployee.java,
щоб знайти застарілий елемент та код, що до нього звертається

Типи анотацій. @SuppressWarnings

- Анотації @SuppressWarnings корисні для пригнічення deprecation- або unchecked-попереджень за допомогою аргументу "deprecation" чи "unchecked".
  - Неусувні (Unchecked) попередження трапляються при змішуванні коду з дженериками та легасі-коду до появи дженериків.
  - Наприклад, у наступному лістингу використовується @SuppressWarnings з аргументом "deprecation" для пригнічення попереджень про застарілість від компілятора, коли код в методі main() класу UseEmployee намагається отримати доступ до поля name класу Employee.

Зауважте! З точки зору стилю програмування, потрібно завжди записувати @SuppressWarnings на найбільш вкладеному елементі, для якого анотація ефективна.

- Наприклад, якщо бажаєте пригнічувати warning від конкретного методу, слід анотувати сам метод, а не його клас.

```
public class UseEmployee
{
   @SuppressWarnings("deprecation")
   public static void main(String[] args)
   {
      Employee emp = new Employee();
      emp.name = "John Doe";
   }
}
```

Оголошення анотованих типів та анотований первинний код

- Перед анотуванням вихідного коду потрібні анотовані типи, які можна інстанціювати.
  - Java постачає багато анотованих типів, крім Override, Deprecated та SuppressWarnings.
- Java також дозволяє оголошувати власні типи за допомогою зарезервованого слова interface.
  - Наприклад, оголосимо анотований тип Stub.
  - Якщо тіло порожнє, екземпляри такого типу називають маркерними анотаціями

```
public @interface Stub
{
}
```

- Приклад використання анотації @Stub – використовується для позначення порожніх методів (stubs).

```
public class Deck // Describes a deck of cards.
{
   @Stub
   public void shuffle()
   {
      // This method is empty and will presumably
      // code at some later date.
   }
}
```

- Хоч здається, що маркерні інтерфейси можна замінити на маркерні анотації, це не так, оскільки маркерні інтерфейси мають свої переваги.
  - Маркерний інтерфейс задає тип, що реалізується за допомогою marked class, що дозволяє перехоплювати проблеми при компіляції.
  - Наприклад, коли клас не реалізує інтерфейс Cloneable, його екземпляри не можуть поверхнево клонуватись за допомогою методу clone() класу Object.
  - Якби Cloneable реалізували як маркерну анотацію, до запуску додатку цю проблему не знайшли б.

Додавання 3 елементів до Stub Annotation Type

- Під елементом розуміють заголовок методу, що з'являється в тілі анотованого типу.
  - Він не може мати параметрів чи викидати clause, а тип, що ним повертається, має бути примітивним, java.lang.String, java.lang.Class, переліченням, анотованим типом або масивом з вище зазначених типів.
  - Проте він може мати значення за умовчанням.

```
public @interface Stub
{
   int id(); // A semicolon must terminate an element declaration.
   String dueDate();
   String developer() default "unassigned";
}
```

Елемент id() задає 32-бітне ціле число, що ідентифікує stub.
Елемент dueDate() встановлює дату в рядковій формі, яке ідентифікує, коли stub метод має реалізовуватись.
В кінці developer() задає рядковий name розробника, що відповідає за код method stub.

Ініціалізація елементів екземпляру Stub

```
public class Deck
{
   @Stub
   (
      id = 1,
      dueDate = "12/21/2012"
   )
   public void shuffle()
   {
   }
}
```

- На відміну від id() та dueDate(), developer() оголошується із значенням за умовчанням "unassigned".
  - Коли ви інстанціюєте Stub та не присвоюєте значення в developer() in that instance, присвоюється дане значення.
- Маємо одну анотацію @Stub, яка ініціалізує свій елемент id() значенням 1, а dueDate() - "12/21/2012".
  - Кожна назва елементу не має trailing(), і список з 2 елементів, розділених комою, з'являється в дужках ().

```
public class Deck
{
   @Stub(value = "1,12/21/2012,unassigned")
   public void shuffle()
   {
   }

   @Stub("2,12/21/2012,unassigned")
   public Card[] deal(int ncards)
   {
      return null;
   }
}
```

- Припустимо, вирішили замінити елементи id(), dueDate(), developer() зі Stub на один елемент String value(), чий рядок задає розділені комами значення ID, due date та developer.
  - Лістинг показує 2 способи ініціалізації value.
  - Коли це лише елемент анотованого типу, можна опустити назву value() та присвоєнння (=) в ініціалізаторі.

Використання мета-анотацій в оголошеннях анотованого типу

- Кожен з анотованих типів (Override, Deprecated, SuppressWarnings) є сам по собі анотованим з мета-анотаціями (анотаціями, що анотують анотовані типи).

```
@Target(value={TYPE,FIELD,METHOD,PARAMETER,CONSTRUCTOR,LOCAL_VARIABLE})
@Retention(value=SOURCE)
public @interface SuppressWarnings
```

- Анотований тип Target, який знаходиться в пакеті java.lang.annotation, ідентифікує види елементів додатку, що яких застосовується анотований тип.
  - @Target показує, що анотації @SuppressWarnings можна використовувати для анотування типів, полів, методів, параметрів, конструкторів та локальних змінних.
  - TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, and LOCAL_VARIABLE є членами перерахування ElementType, яке також міститься в пакеті java.lang.annotation.

Анотований тип Retention (пакет java.lang.annotation)

- Ідентифікує retention (також lifetime – тривалість життя) анотацій анотованого типу.
  - @Retention вказує на те, що анотації @SuppressWarnings мають тривалість життя, обмежену первинним кодом—вони не існують після компіляції.
  - ***SOURCE*** – один з членів перелічення RetentionPolicy (пакет java.lang.annotation). Компілятор відкидає анотації після їх використання.
  - ***CLASS***: компілятор записує анотації в class-файл, проте віртуальна машина не retain їх (щоб зберегти пам’ять). Політика за замовчуванням.
  - ***RUNTIME***: компілятор записує анотації в class-файл, а віртуальна машина retain їх, щоб вони могли зчитуватись Reflection API під час виконання.
- Існують 2 проблеми з анотованим типом Stub.
  - Перша з них – відсутність мета-анотації @Target – означає, що можна анотувати будь-який елемент додатку за допомогою @Stub.

Анотуємо небажані елементи додатку

```
@Stub("1,12/21/2012,unassigned")
public class Deck
{
   @Stub("2,12/21/2012,unassigned")
   private Card[] cardsRemaining = new Card[52];
   @Stub("3,12/21/2012,unassigned")
   public Deck()
   {
   }

   @Stub("4,12/21/2012,unassigned")
   public void shuffle()
   {
   }

   @Stub("5,12/21/2012,unassigned")
   public Card[] deal(@Stub("5,12/21/2012,unassigned") int ncards)
   {
      return null;
   }
}
```

- Використовуємо @Stub, щоб анотувати клас Deck, поле cardsRemaining та параметр ncards разом з конструктором і 2 методами.
  - Перші 3 елементи додатку не підходять для анотування, оскільки не є заглушками (stubs).

Анотуємо небажані елементи додатку

- Можна поправити проблему, дописуючи перед оголошеннями анотованого типу Stub @Target({ElementType.METHOD, ElementType.CONSTRUCTOR}), щоб Stub застосовувався лише до методів та конструкторів.
  - Після цього компілятор виведе наступні помилки:

```
Deck.java:1: error: annotation type not applicable to this kind of declaration
@Stub("1,12/21/2012,unassigned")
^
Deck.java:4: error: annotation type not applicable to this kind of declaration
   @Stub("2,12/21/2012,unassigned")
   ^
Deck.java:18: error: annotation type not applicable to this kind of declaration
   public Card[] deal(@Stub("5,12/21/2012,unassigned") int ncards)
                      ^
3 errors
```

Налагоджений анотований тип Stub

- Друга проблема: CLASS retention policy за умовчанням робить неможливим обробити анотації @Stub підчас виконання.
  - Вирішується дописуванням префіксу @Retention(RetentionPolicy.RUNTIME) для Stub

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
```

```
@Target({ElementType.METHOD, ElementType.CONSTRUCTOR})
@Retention(RetentionPolicy.RUNTIME)
public @interface Stub
{
   String value();
}
```