MODUL PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

ANNOTATION, GENERICS, LAMBDA EXPRESSION, METHOD REFERENCE

Deskripsi Singkat

Praktikum pemrograman berorientasi objek adalah praktikum yang menggunakan bahasa Java sebagai bantuan dalam memahami konsep pemrograman berorientasi objek. Materi praktikum berisi teori, latihan dan soal pemrograman.

Tujuan

1. Memahami penggunaan annotation.
2. Memahami penggunaan generics.
3. Memahami penggunaan lambda expression.
4. Memahami penggunaan method reference.

Prasyarat

Siswa telah melakukan praktikum 1-11.

Materi 1 : Annotation

Annotation merupakan informasi data tentang kode program tetapi tidak akan berdampak secara langsung pada kode. Annotation berupa label atau penanda berformat metadata yang dimasukkan ke dalam class, interface, method atau field/variable. Annotation mengindikasikan informasi tambahan yang digunakan oleh Java compiler dan JVM.

Annotation dalam bentuk yang paling sederhana:

```
@Entity
```

Simbol at (@) akan memberitahu compiler bahwa yang mengikuti setelah tanda tersebut adalah annotation.

Materi 2 : Generics

Java generics digunakan untuk mendeteksi bugs pada saat kompilasi dan bugs saat runtime. Generics diberikan symbol <E> atau <T>. Generics umumnya digunakan pada class-class yang ada pada Java Collection Framework.

Sintaks:

```
ClassAtauInterface<Type>
```

Contoh:

```
ArrayList<String>
```

Materi 3 : Lambda expression

Lambda expression menyediakan implementasi bagi *functional interface*. Interface yang hanya memiliki satu method abstract disebut sebagai functional interface. Java menyediakan annotation *@FunctionalInterface*, yang dapat digunakan untuk mendeklarasikan interface sebagai functional interface.

Lambda expression dapat menyingkat code. Sebab dengan menggunakan lambda expression, kita tidak perlu mendefinisikan method lagi namun langsung menuliskan kode implementasi dari method abstract pada interface.

Sintaks:

```
(argument-list) -> {body}
```

Lambda expression pada Java terdiri dari 3 komponen:

1. Argument-list: boleh kosong atau diisi dengan argument
2. Arrow-token: digunakan untuk mengaitkan argument-list dengan bagian body
3. Body: berisi ekspresi dan statement untuk lambda expression

Materi 4 : Method reference

Method reference digunakan untuk merujuk/me-*refer* method pada functional interface. Method reference merupakan bentuk yang lebih ringkas dan mudah dari lambda expression. Jadi setiap kali kita menggunakan lambda expression untuk merujuk method maka lambda expression tersebut dapat diganti dengan method reference.

Terdapat 3 tipe method reference:

1. Reference ke method static
2. Reference ke method instance
3. Reference ke constructor

Contoh untuk ketiga method reference tersebut dapat dilihat pada bagian latihan.

LATIHAN 1

Ketik dan simpan coding di bawah dengan nama file Mutator1_value.java

```
public @interface Mutator1_value {
    String value();
}
```

Ketik dan simpan coding di bawah dengan nama file Mutator2_variable.java

```
public @interface Mutator2_variable {
    String variable();
}
```

Ketik dan simpan coding di bawah dengan nama file Accessor.java

```
public @interface Accessor {
    String variableName();
    String variableType() default "String";;
}
```

Ketik dan jalankan coding di bawah:

```
public class Main {

    private String name;
    private int id;

      public Main(){
            name = "Java";
      }

    @Mutator1_value("xyz")
    public void setName(String name) {
        this.name = name;
    }

    @Mutator2_variable(variable = "xyz")
    public int setId(int id) {
        this.id = id;
    }

      @Accessor(variableName = "name", variableType = "int")
    public int getId() {
        return id;
    }

      // "variableType" bisa tidak dituliskan karenn memiliki default
value
```

```
    // Namun jika tidak ada default value maka harus dituliskan secara
eksplisit
    @Accessor(variableName = "xyz")
    public String getName() {
        return name;
    }

      @Override
    public boolean equals(Object otherName) {
        String newName = (String) otherName;
        int comparison = name.compareTo(newName);

        return (comparison == 0);
    }

    public static void main(String[] args) {
        System.out.println("No compiler error");
    }

}
```

Apakah hasil outputnya ketika coding di atas dijalankan?

Kemudian ubah bagian :

```
@Override
    public boolean equals(Object otherName) {
        String newName = (String) otherName;
        int comparison = name.compareTo(newName);

        return (comparison == 0);
    }
```

Menjadi:

```
@Override
    public boolean equals(String otherName) {
        String newName = (String) otherName;
        int comparison = name.compareTo(newName);

        return (comparison == 0);
    }
```

Kompilasi program yang telah diubah. Apakah error yang muncul?

Error tersebut muncul karena penggunaan annotation @Override. @Override digunakan pada method overriding. Sebab dengan adanya @Override, compiler akan mengecek apakah superclass memiliki method equals (String otherName), jika tidak ada maka muncul error.

LATIHAN 2

Ketik dan jalankan coding di bawah:

```
import java.util.*;

class TestGenerics1{
       public static void main(String args[]){
              ArrayList<String> list=new ArrayList<String>();
              list.add("rani");
              list.add("rijal");
              //list.add(32);  //compile time error

              String s=list.get(1); //tidak perlu melakukan casting
              System.out.println("element is: "+s);

              Iterator<String> itr=list.iterator();
              while(itr.hasNext()){
                     System.out.println(itr.next());
              }
       }
}
```

Apakah hasil outputnya ketika coding di atas dijalankan?

Coba anda hilangkan symbol komentar pada

```
       //list.add(32);  //compile time error
```

Menjadi

```
       list.add(32);  //compile time error
```

Kompilasi program tersebut. Apakah error yang muncul?

Hal tersebut muncul sebab dengan penggunaan generics pada code ArrayList<String> list maka kita ingin memastikan bahwa data yang disimpan di dalam variable list bertipe data String saja. Sehingga jika ada tipe data lain, maka akan muncul error.

LATIHAN 3

Ketik dan jalankan coding di bawah:

```
interface Drawable{
    public void draw();
}

public class LambdaExpressionExample {
```

```
    public static void main(String[] args) {
        int width=10;

        //tanpa adanya lambda expression,
           //implementasi Drawable menggunakan anonymous inner class

        Drawable d=new Drawable(){
            public void draw(){
                      System.out.println("Drawing "+width);
                 }
        };
        d.draw();
    }
}
```

Apakah hasil outputnya ketika coding di atas dijalankan? Pahami output yang terhasil.

Kemudian kita coba menggunakan lambda expression.

```
@FunctionalInterface  //annotation ini optional
interface Drawable{
    public void draw();
}

public class LambdaExpressionExample2 {
    public static void main(String[] args) {
        int width=10;

        //dengan lambda expression
        Drawable d2=()->{
            System.out.println("Drawing "+width);
        };
        d2.draw();
    }
}
```

Kompilasi dan jalankan program. Outputnya tetap sama, namun dari sisi code program menjadi lebih singkat.

LATIHAN 4

Ketik dan jalankan coding di bawah:

```
interface Sayable{
    public String say();
}

public class LambdaExpressionExample3{
     public static void main(String[] args) {
```

```
            Sayable s=()->{
                  return "I have nothing to say.";
            };
            System.out.println(s.say());
      }
}
```

Apakah hasil outputnya ketika coding di atas dijalankan? Pahami output yang terhasil.

Argument-list pada lambda expression dapat memiliki argument atau tidak ada argument. Pada contoh di atas, tidak ada argument. Contoh di bawah akan menampilkan lambda expression yang memiliki argument pada argument-list.

```
interface Sayable{
    public String say(String name);
}

public class LambdaExpressionExample4{
    public static void main(String[] args) {

        // Lambda expression with single parameter.
        Sayable s1=(name)->{
            return "Hello, "+name;
        };
        System.out.println(s1.say("Sana"));

        // You can omit function parentheses
        Sayable s2= name ->{
            return "Hello, "+name;
        };
        System.out.println(s2.say("Sini"));
    }
}
```

Contoh berikut merupakan lambda expression yang memiliki dua parameter sebagai argument-list.

```
interface Addable{
    int add(int a,int b);
}

public class LambdaExpressionExample5{
    public static void main(String[] args) {

        // Multiple parameters in lambda expression
        Addable ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));

        // Multiple parameters with data type in lambda expression
        Addable ad2=(int a,int b)->(a+b);
        System.out.println(ad2.add(100,200));
```

```
    }
}
```

LATIHAN 5

Contoh berikut ini adalah method reference ke method static. Ketik dan jalankan coding di bawah:

```
interface Sayable{
    void say();
}
public class MethodReference {
    public static void saySomething(){
        System.out.println("Hello, this is static method.");
    }
    public static void main(String[] args) {
        // Referring static method
        Sayable sayable = MethodReference::saySomething;

        // Calling interface method
        sayable.say();
    }
}
```

Apakah hasil outputnya ketika coding di atas dijalankan? Pahami output yang terhasil.

Contoh berikut ini adalah method reference ke method instance. Ketik dan jalankan coding di bawah:

```
interface Sayable{
    void say();
}
public class InstanceMethodReference {
    public void saySomething(){
        System.out.println("Hello, this is non-static method.");
    }
    public static void main(String[] args) {
           // Creating object
        InstanceMethodReference       methodReference       =       new
InstanceMethodReference();

        // Referring non-static method using reference
        Sayable sayable = methodReference::saySomething;

           // Calling interface method
        sayable.say();

           // Referring non-static method using anonymous object
        Sayable               sayable2               =               new
InstanceMethodReference()::saySomething;
```

```
            // Calling interface method
            sayable2.say();
    }
}
```

Apakah hasil outputnya ketika coding di atas dijalankan? Pahami output yang terhasil.

Contoh berikut ini adalah method reference ke constructor. Ketik dan jalankan coding di bawah:

```
interface Messageable{
    Message getMessage(String msg);
}

class Message{
    Message(String msg){
        System.out.print(msg);
    }
}

public class ConstructorReference {
    public static void main(String[] args) {
        Messageable hello = Message::new;
        hello.getMessage("Hello");
    }
}
```

Apakah hasil outputnya ketika coding di atas dijalankan? Pahami output yang terhasil.

SOAL-SOAL

1. .