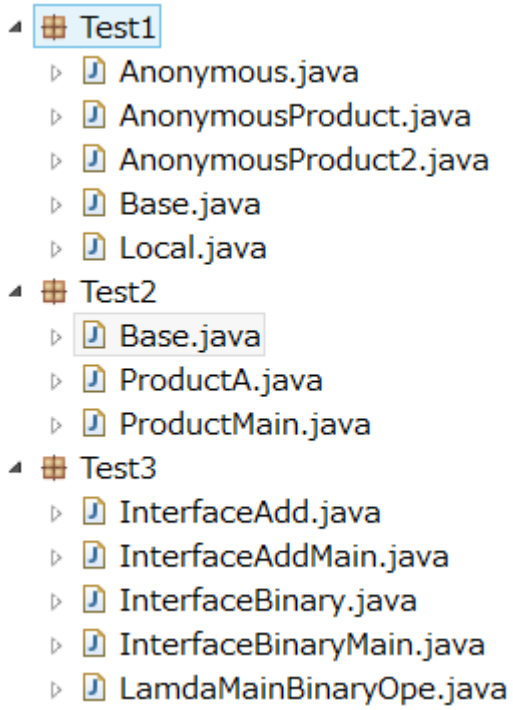
Index

1-1. Local Class
1-2. Super class , Sub class, Override
1-3. Abstract class
1-4. Interface

2-1. Abstract + Anonymous class
2-2. Abstract + Anonymous class
2-3. Object class + Anonymous Class
2-4. Interface + Anonymous class

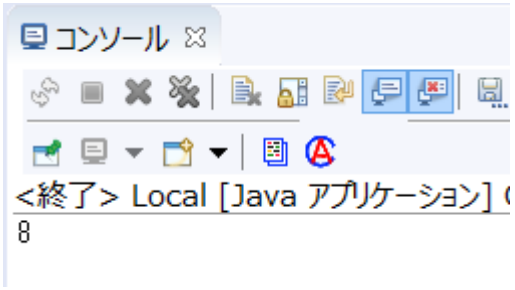
3-1. Lamda
3-2. Lamda



1-1. Local Class

In the **main method**,
① define class LocalTest1, ② instantation of LocalTest1, ③ call method f.

```
package Test1;

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

 class LocalTest1{                    ①
 public int f(int a, int b) {
 return a + b;
 }
 }

LocalTest1 l1 = new LocalTest1();     ②
System.out.println(l1.f(3, 5));
}                    ③
}
```



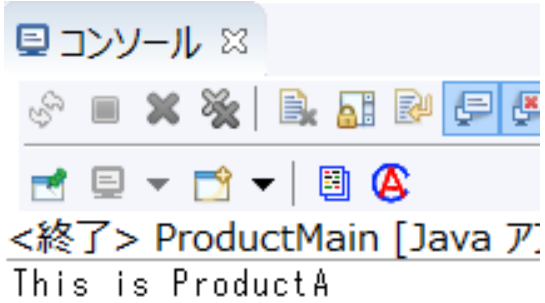
1-2. Super class , Sub class, Override

https://docs.oracle.com/javase/8/docs/api/java/lang/Override.html

Sperclass method is overwritten by Subclass method.

Super

```
package Test2;

class base{
  public void dispName(){
    System.out.println
("Alert:Please define ProductName in sub class");
  } }
```



Sub

```
package Test2;

public class ProductA extends base {
@Override
  public void dispName(){
    System.out.println("This is ProductA");
  };
}
```

Main

```
package Test2;

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

ProductA productname = new ProductA();
productname.dispName();
} }
```

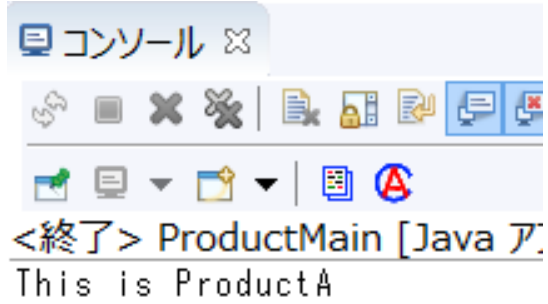
1-3. Abstract class

https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

An abstract class is a class that is declared abstract—it may or may not include abstract methods.
Abstract classes cannot be instantiated, but they can be subclassed.
An abstract method is a method that is declared without an implementation.

Abstract

```
package Test2;

abstract class base{
abstract public void dispName();
}
```



Sub

```
package Test2;

public class ProductA extends base {

public void dispName(){
    System.out.println("This is ProductA");
  };
}
```

Main

```
package Test2;

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

ProductA productname = new ProductA();
productname.dispName();
} }
```

1-4. Interface

https://www.w3schools.com/java/java_interface.asp

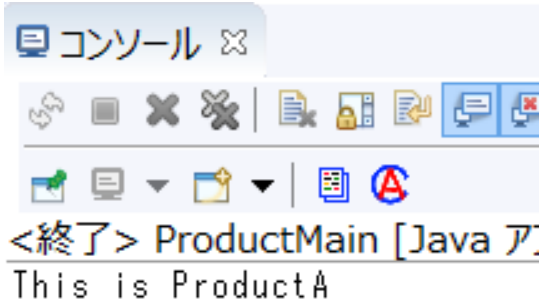
Another way to achieve abstraction in Java, is with interfaces.
An interface is a completely "abstract class" that is used to group related methods with empty bodies:.

Interface

```
package Test2;

interface  base{
 void dispName();
}
```

Sub

```
package Test2;

class ProductA implements base {
@Override
  public void dispName(){
    System.out.println("This is ProductA");
  };
}
```



Main

```
package Test2;

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

ProductA productname = new ProductA();
productname.dispName();
}
}
```

2-1. Abstract + Anonymous class

Example-1

https://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html#:~:text=Anonymous%20classes%20enable%20you%20to,a%20local%20class%20only%20once.

Anonymous classes enable you to make your code more concise.
They enable you to **declare a class** and **instantiate a class** at the same time.
They are like local classes except that they do not have a name. Use them if **you need to use a local class only once**.

Abstract

```
package Test1;

abstract class base{
abstract public void dispName();
}
```

There is not Subclass

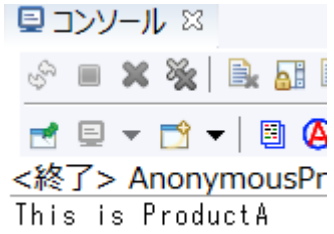
```
package Test1;

public class ProductA extends base {

public void d
    System.ou
  };
}
```

Anonymous class means…
actually instantation for
“class Anonymous extends base”

Main

① instantation
② override method and define method
③ call method

```
package Test1;

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

base producta = new base(){                    ①
@Override
public void dispName(){                        ②
System.out.println("This is ProductA");
 }
};
producta.dispName();
} object name. method name                     ③
}
```



2-2. Abstract + Anonymous class

Example-2

Anonymous classes enable you to make your code more concise.
They enable you to **declare a class** and **instantiate a class** at the same time.
They are like local classes except that they do not have a name. Use them if **you need to use a local class only once**.

Abstract

```
package Test1;

abstract class base{
abstract public void dispName();
}
```

There is not Subclass

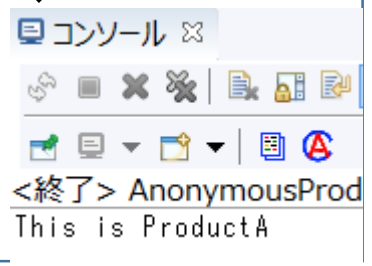
```
package Test1;

public class ProductA extends base {

public void d
    System.ou
  };
}
```

Anonymous class means…
actually instantation for
“class Anonymous extends base”

Main

① instantation of class w/o object name
② override method and define method
③ call method

```
package Test1;

public class AnonymousProduct {
public static void main(String[] args) {
      object name
new base(){
@Override          ①
public void dispName(){                ②
System.out.println("This is ProductA");
 }
}.dispName();③
}  . method name
}
```



2-3. Object class + Anonymous Class

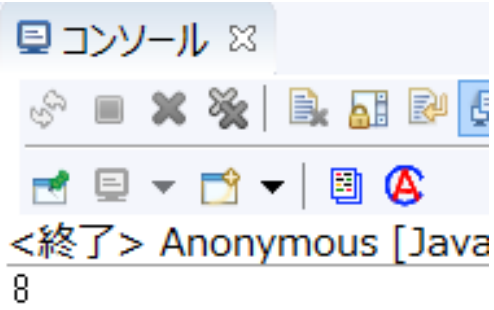
Example-3

Anonymous classes enable you to make your code more concise.
They enable you to **declare a class** and **instantiate a class** at the same time.
They are like local classes except that they do not have a name. Use them if **you need to use a local class only once**.

```
package Test1;

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

int x = new Object() {
        public int f(int a, int b) {
        return a + b;
}
}.f(3, 5);

System.out.println(x);
}

}
```

Class Object is the root of the class hierarchy. Every class has Object as a superclass.

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

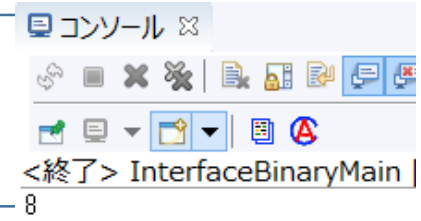


2-4. Interface + Anonymous class

Example-4

https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/lambda-quickstart/index.html

Another way to achieve abstraction in Java, is with interfaces.
An interface is a completely "abstract class" that is used to group related methods with empty bodies.



Interface

```
package Test3;

interface TestBinary {

   public int inttest(int left, int right);
}
```

① Instantation
② define **inttest** method and return
③ call method with argument int a & int b

Main

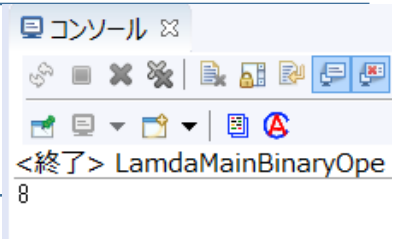
```
package Test3;

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

TestBinary test = new TestBinary(){    ①

public int inttest(int a, int b){      ②
return a + b;
 }
};

System.out.println(test.inttest(3,5)); ③
}
}
```

3-1. Lamda

Example-1

https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/lambda-quickstart/index.html

Lambda expressions are a new and important feature included in Java SE 8.
They provide a clear and concise way to represent one method interface using an expression.



Interface

```
package Test3;

interface AddTest{

    public int f(int a);
}
```

Main

① instantation (w/o new)
② (type, argument) in f method in Interface
③ define method and return
④ call method with argument int a & int b

```
package Test3;

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

AddTest i1 = (int a)->{

return a + 5;
};
System.out.println(i1.f(10));
}
}
```

① ② ③ ④

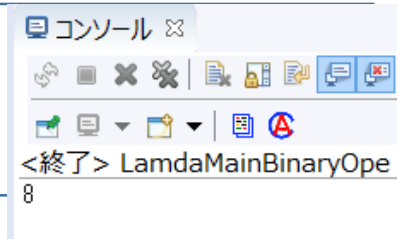
object name. method name

3-2. Lamda

Example-2

https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/lambda-quickstart/index.html

Lambda expressions are a new and important feature included in Java SE 8.
They provide a clear and concise way to represent one method interface using an expression.



Functional interface in Java

MainBinaryOpe.java | IntBinaryOperator.class [FernFlower]

```
 2 * Copyright (c) 2012, 2013, Oracle and/or its affiliates.
25 package java.util.function;
26
27 /**
28  * Represents an operation upon two {@code int}-valued oper
29  * {@code int}-valued result.   This is the primitive type
30  * {@link BinaryOperator} for {@code int}.
31  *
32  * <p>This is a <a href="package-summary.html">functional i
33  * whose functional method is {@link #applyAsInt(int, int)}
34  *
35  * @see BinaryOperator
36  * @see IntUnaryOperator
37  * @since 1.8
38  */
39 @FunctionalInterface
40 public interface IntBinaryOperator {
41
42     /**
43      * Applies this operator to the given operands.
44      *
45      * @param left the first operand
46      * @param right the second operand
47      * @return the operator result
48      */
49     int applyAsInt(int left, int right);
50 }
```

Main

① instantation (w/o new)
② (type, argument) in applyAsInt method in Interface
③ define method and return
④ call method with argument int a & int b

```
package Test3;
import java.util.function.IntBinaryOperator;

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

IntBinaryOperator i1 = (int a, int b)->{
return a + b;
};

System.out.println(i1.applyAsInt(3, 5));
} }
```

① ② ③ ④

④ object name. method name