[DefaultMethodDemo](#_lpfnih94hfze)

[LambdaDemo](#_wda0qqiaaohj)

[LambdaDemo2](#_2edd35651ujs)

[LambdaDemo3](#_2bxe8kfs2qer)

[MethodReferenceDemo](#_cfzz932gmzob)

[MethodReferenceDemo2](#_k59ucgr34tj6)

[MethodReferenceDemo3](#_4nh60hx6o1u)

[MethodReferenceDemo4](#_77ws3sai0pab)

[MethodReferenceDemo5](#_bq0e30ma13r7)

[ConstructorReferenceDemo](#_syeavl7k9sg6)

[ConstructorReferenceDemo2](#_6hxh499aokbb)

[PredicateDemo](#_oq6jzgm203a0)

[FunctionDemo](#_926kyt160tjv)

[ConsumerDemo](#_yt3ale35l2j8)

[SupplierDemo](#_qnysul6ffyux)

# 

# 

# DefaultMethodDemo

|  |
| --- |
| public class DefaultMethodDemo {  interface G {  void m();  default void m2() {  System.out.println("G#m2");  }  }  public static void main(String[] args) {  G g = new G() {  public void m() {  System.out.println("Anonymous#m");  }  };  g.m();  g.m2();  }  } |

|  |
| --- |
| Anonymous#m  G#m2 |

# LambdaDemo

|  |
| --- |
| public class LambdaDemo {  interface G {  void m();  }  public static void main(String[] args) {  G g;  // (1)  g = () -> {  System.out.println("Lambda impl G#m");  };  g.m();  // (2)  g = () -> System.out.println("Lambda impl G#m");  g.m();    // You can use an anonymous class for this purpose  g = new G() {  public void m() {  System.out.println("Anonymous#m");  }  };  g.m();  }  } |

|  |
| --- |
| Lambda impl G#m  Lambda impl G#m  Anonymous#m |

# LambdaDemo2

|  |
| --- |
| public class LambdaDemo2 {  interface G {  void m(Object x);  }  public static void main(String[] args) {  G g;    // (1)  g = (Object x) -> {  System.out.println("Lambda impl G#m(Object) ==> " + x);  };  g.m("ABC");  // (2)  g = (x) -> {  System.out.println("Lambda impl G#m(Object) ==> " + x);  };  g.m("ABC");  // (3)  g = (x) -> System.out.println("Lambda impl G#m(Object) ==> " + x);  g.m("ABC");  // You can use an anonymous class for this purpose  g = new G() {  public void m(Object x) {  System.out.println("Anonymous#m ==> " + x);  }  };  g.m("ABC");  }  } |

|  |
| --- |
| Lambda impl G#m(Object) ==> ABC  Lambda impl G#m(Object) ==> ABC  Lambda impl G#m(Object) ==> ABC  Anonymous#m ==> ABC |

# LambdaDemo3

|  |
| --- |
| public class LambdaDemo3 {  interface G {  Object m(Object x);  }  public static void main(String[] args) {  G g;  // (1)  g = (Object x) -> {  return "Lambda impl G#m(Object) ==> " + x;  };  System.out.println(g.m("ABC"));  // (2)  g = (Object x) -> "Lambda impl G#m(Object) ==> " + x;  System.out.println(g.m("ABC"));  // (3)  g = (x) -> "Lambda impl G#m(Object) ==> " + x;  System.out.println(g.m("ABC"));    // You can use an anonymous class for this purpose  g = new G() {  public Object m(Object x) {  return "Anonymous#m ==> " + x;  }  };  System.out.println(g.m("ABC"));  }  } |

|  |
| --- |
| Lambda impl G#m(Object) ==> ABC  Lambda impl G#m(Object) ==> ABC  Lambda impl G#m(Object) ==> ABC  Anonymous#m ==> ABC |

# MethodReferenceDemo

|  |
| --- |
| public class MethodReferenceDemo {  interface G {  void m(Object x);  }  static class Util {  static void m2(Object x) {  System.out.println("Util#m2 ==> " + x);  }  }  public static void main(String[] args) {  G g;  // (1)  g = Util::m2;  g.m("ABC");  // You can use an anonymous class for this purpose  g = new G() {  public void m(Object x) {  Util.m2(x);  }  };  g.m("ABC");  }  } |

|  |
| --- |
| Util#m2 ==> ABC  Util#m2 ==> ABC |

# MethodReferenceDemo2

|  |
| --- |
| public class MethodReferenceDemo2 {  interface G {  void m(Object x, Object y);  }  static class Util {  static void m2(Object x, Object y) {  System.out.println("Util#m2 ==> " + x + ", " + y);  }  }  public static void main(String[] args) {  G g;  // (1)  g = Util::m2;  g.m("ABC", "DEF");  // You can use an anonymous class for this purpose  g = new G() {  public void m(Object x, Object y) {  Util.m2(x, y);  }  };  g.m("ABC", "DEF");  }  } |

|  |
| --- |
| Util#m2 ==> ABC, DEF  Util#m2 ==> ABC, DEF |

# MethodReferenceDemo3

|  |
| --- |
| public class MethodReferenceDemo3 {  interface G {  void m(Object x, Object y);  }  static class SomeClass {  void m2(Object x, Object y) {  System.out.println("SomeClass#m2 ==> " + x + ", " + y);  }  }  public static void main(String[] args) {  G g;  SomeClass sc = new SomeClass();  // (1)  g = sc::m2;  g.m("ABC", "DEF");  // You can use an anonymous class for this purpose  g = new G() {  public void m(Object x, Object y) {  sc.m2(x, y);  }  };  g.m("ABC", "DEF");  }  } |

|  |
| --- |
| SomeClass#m2 ==> ABC, DEF  SomeClass#m2 ==> ABC, DEF |

# MethodReferenceDemo4

|  |
| --- |
| public class MethodReferenceDemo4 {  interface G {  char getChar(int x);  }  public static void main(String[] args) {  G g;  // (1)  g = "asdf"::charAt;  System.out.println(g.getChar(3));  // You can use an anonymous class for this purpose  g = new G() {  public char getChar(int x) {  return "asdf".charAt(x);  }  };  System.out.println(g.getChar(3));  }  } |

|  |
| --- |
| f  f |

# 

# MethodReferenceDemo5

|  |
| --- |
| public class MethodReferenceDemo5 {    interface G {  int getInt(String s);  }  public static void main(String[] args) {  G g;  int x;  // (1)  g = Integer::parseInt;  x = g.getInt("345");  System.out.println(x);    // You can use an anonymous class for this purpose  g = new G() {  public int getInt(String s) {  return Integer.parseInt(s);  }  };  x = g.getInt("345");  System.out.println(x);  }  } |

|  |
| --- |
| 345  345 |

# ConstructorReferenceDemo

|  |
| --- |
| public class ConstructorReferenceDemo {  interface H {  Object create();  }    static class SomeClass {  SomeClass() {  System.out.println("SomeClass#constr");  }  }  public static void main(String[] args) {  H h;  // (1)  h = SomeClass::new;  h.create();  // You can use an anonymous class for this purpose  h = new H() {  public Object create() {  return new SomeClass();  }  };  h.create();  }  } |

|  |
| --- |
| SomeClass#constr  SomeClass#constr |

# ConstructorReferenceDemo2

|  |
| --- |
| public class ConstructorReferenceDemo2 {  interface H {  String create(byte[] bytes);  }  public static void main(String[] args) {  H h;  byte[] bytes = {65, 66, 67};  // (1)  h = String::new;  System.out.println(h.create(bytes));  // You can use an anonymous class for this purpose  h = new H() {  public String create(byte[] bytes) {  return new String(bytes);  }  };  System.out.println(h.create(bytes));  }  } |

|  |
| --- |
| ABC  ABC |

# PredicateDemo

|  |
| --- |
| import java.lang.reflect.Modifier;  import java.util.concurrent.atomic.AtomicInteger;  import java.util.function.Predicate;  public class PredicateDemo {  public static void main(String[] args) {  Predicate isCharacter = (x) -> x instanceof Character;  Predicate isBoolean = (x) -> x instanceof Boolean;  Predicate isNumber = (x) -> x instanceof Number;  Predicate isFinalType = (x) ->  Modifier.isFinal(x.getClass().getModifiers());  Predicate isWrapper = isCharacter.or(isBoolean)  .or(isNumber.and(isFinalType));  System.out.println(isWrapper.test(3));  System.out.println(isWrapper.test(3.5F));  System.out.println(isWrapper.test('a'));  System.out.println(isWrapper.test(new AtomicInteger(0)));  System.out.println(isWrapper.test("asdf"));  }  } |

|  |
| --- |
| true  true  true  false  false |

|  |
| --- |
| package java.util.function;  @FunctionalInterface  public interface Predicate<T> {  boolean test(T t);  ...  } |

# FunctionDemo

|  |
| --- |
| import java.util.function.Function;  public class FunctionDemo {  public static void main(String[] args) {  Function a = (x) -> x + "A";  Function b = (x) -> x + "B";  Function c = (x) -> x + "C";  Function d = (x) -> x + "D";  System.out.println(a.compose(b).andThen(c).apply("str"));  // str + "B" + "A" + "C"  System.out.println(a.andThen(b).compose(c).compose(d).apply("str"));  // str + "D" + "C" + "A" + "B"  }  } |

|  |
| --- |
| strBAC  strDCAB |

|  |
| --- |
| package java.util.functions;  @FunctionalInterface  public interface Function<T, R> {  R apply(T t);  ...  } |

# ConsumerDemo

|  |
| --- |
| import java.util.function.Consumer;  public class SupplierDemo {    public static void main(String[] args) {  Consumer printClassName = (x) ->  System.out.println(x.getClass().getName());  Consumer printSuperclassName = (x) ->  System.out.println(x.getClass().getSuperclass().getName());  printClassName.andThen(printSuperclassName).accept(7);  }  } |

|  |
| --- |
| java.lang.Integer  java.lang.Number |

# SupplierDemo

|  |
| --- |
| import java.util.function.Supplier;  public class SupplierDemo {    public static void main(String[] args) {  Supplier s;    s = StringBuilder::new;  System.out.println(s.get().getClass());      s = () -> {  class A {}  return new A();  };  System.out.println(s.get().getClass());  }  } |

|  |
| --- |
| class java.lang.StringBuilder  class SupplierDemo$1A |