JAVA

Basic

```
public class FirstJavaProgram {
  public static void main(String[] args){
    System.out.println("This is my first program in java");
  }//End of main
}//End of FirstJavaProgram Class
```

Lambda

```
(syntax of lambda expression)
(parameter_list) -> {function_body}
```

Если лямбда-выражение не имеет параметров, вы все равно ставите пустые скобки, так же, как с методом без параметров:

```
() -> { for (int i = 0; i < 1000; i++) doWork(); }
```

Если типы параметров лямбда-выражения можно вывести, можно опустить их. Например,

```
Comparator<String> comp
    = (firstStr, secondStr) // Same as (String firstStr, String secondStr)
        -> Integer.compare(firstStr.length(),secondStr.length());
```

Здесь компилятор может сделать вывод, что `firstStr` и `secondStr` должны быть строками, потому что лямбда-выражение присваивается компаратору строк. (Мы посмотрим на это присваивание повнимательнее позже.)

Если метод имеет один параметр выводимого типа, вы можете даже опустить скобки:

```
EventHandler<ActionEvent> listener = event ->
    System.out.println("The button has been clicked!");
        // Instead of (event) -> or (ActionEvent event) ->
```

Вы никогда не указываете тип результата лямбда-выражения. Это всегда выясняется из контекста. Например, выражение

```
(String firstStr, String secondStr) -> Integer.compare(firstStr.length(), secondStr.length())
```

Anonymous inner class

```
(HelloWorld is an interface declared before)
HelloWorld frenchGreeting = new HelloWorld() {
    String name = "tout le monde";
    public void greet() {
        greetSomeone("tout le monde");
    }
    public void greetSomeone(String someone) {
        name = someone;
        System.out.println("Salut " + name);
    }
};
```

Template

A *generic type* is a generic class or interface that is parameterized over types. The following `Box` class will be modified to demonstrate the concept.

A *generic class* is defined with the following format:

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

```
example
public class Box<T> {
    // T stands for "Type"
    private T t;
```

```
    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

To reference the generic `Box` class from within your code, you must perform a *generic type invocation*, which replaces `T` with some concrete value, such as `Integer`:

```
Box<Integer> integerBox;
```

In Java SE 7 and later, you can replace the type arguments required to invoke the constructor of a generic class with an empty set of type arguments (<>) as long as the compiler can determine, or infer, the type arguments from the context. This pair of angle brackets, <>, is informally called *the diamond*. For example, you can create an instance of `Box<Integer>` with the following statement:

```
Box<Integer> integerBox = new Box<>();
```

To declare a bounded type parameter, list the type parameter's name, followed by the extends keyword, followed by its *upper bound*, which in this example is Number. Note that, in this context, extends is used in a general sense to mean either "extends" (as in classes) or "implements" (as in interfaces).

```
public class NaturalNumber<T extends Integer> {
BUT in general…
Inheritance
public class MountainBike extends Bicycle {…
Interface
class Demo implements MyInterface {…
```

Virtual

every **non-static method** in JAVA is by default *virtual method* **except final and private methods**. The methods which cannot be inherited for polymorphic behavior is not a virtual method.

Abstract vlass VS interface

	Abstract Class	Interface
1	An abstract class can extend only one class or one abstract class at a time	An interface can extend any number of interfaces at a time
2	An abstract class can extend another concrete (regular) class or abstract class	An interface can only extend another interface
3	An abstract class can have both abstract and concrete methods	An interface can have only abstract methods
4	In abstract class keyword “abstract” is mandatory to declare a method as an abstract	In an interface keyword “abstract” is optional to declare a method as an abstract
5	An abstract class can have protected and public abstract methods	An interface can have only have public abstract methods
6	An abstract class can have static, final or static final variable with any access specifier	interface can only have public static final (constant) variable