Agile Software Development

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

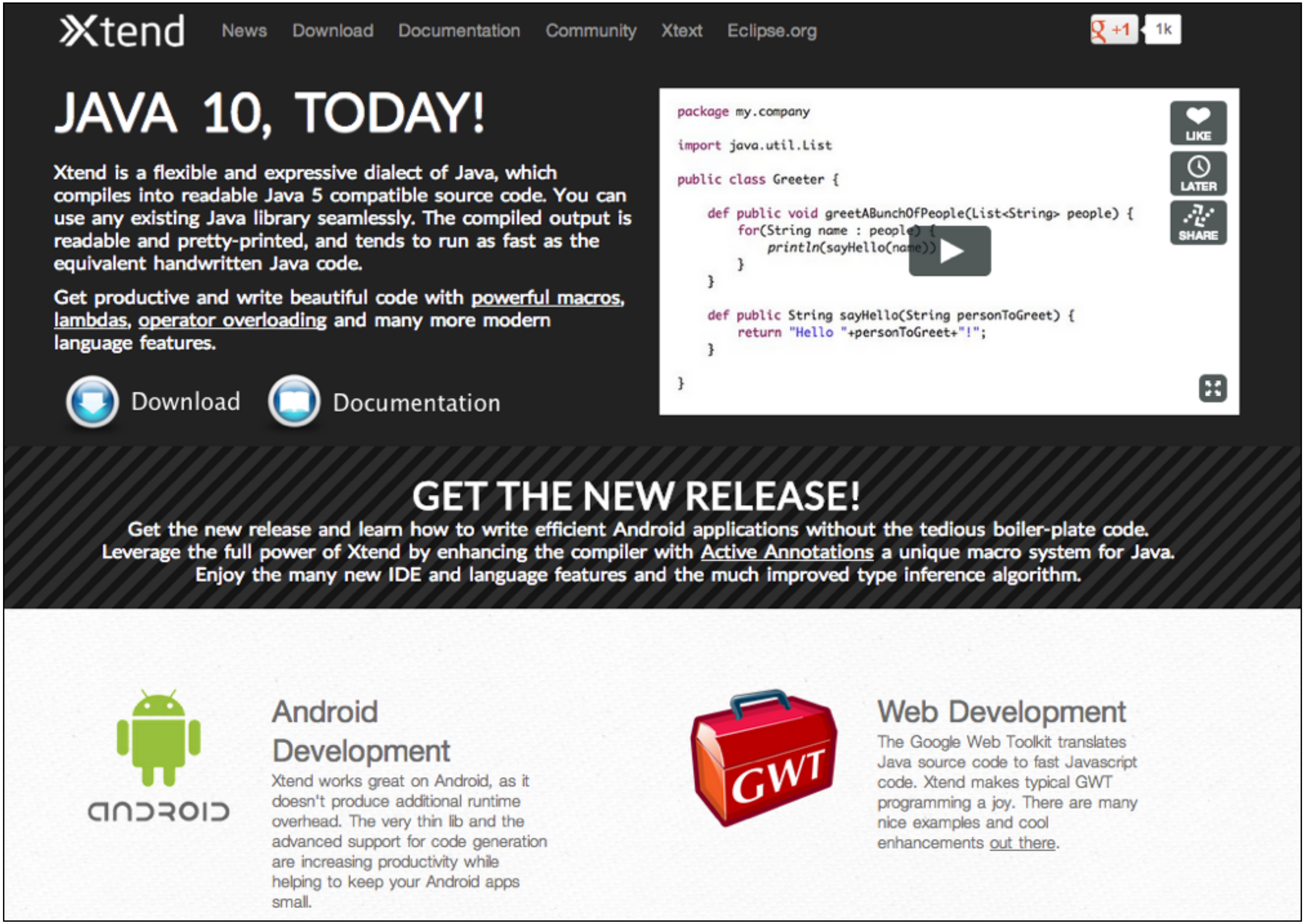
http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Xtend Programming Language



JAVA 10, TODAY!

Xtend is a flexible and expressive dialect of Java, which compiles into readable Java 5 compatible source code. You can use any existing Java library seamlessly. The compiled output is readable and pretty-printed, and tends to run as fast as the equivalent handwritten Java code.

Get productive and write beautiful code with powerful macros, lambdas, operator overloading and many more modern language features.

Download Documentation

```
package my.company

import java.util.List

public class Greeter {

    def public void greetABunchOfPeople(List<String> people) {
        for(String name : people) {
            println(sayHello(name))
        }
    }

    def public String sayHello(String personToGreet) {
        return "Hello "+personToGreet+"!";
    }

}
```

GET THE NEW RELEASE!

Get the new release and learn how to write efficient Android applications without the tedious boiler-plate code.
Leverage the full power of Xtend by enhancing the compiler with Active Annotations a unique macro system for Java.
Enjoy the many new IDE and language features and the much improved type inference algorithm.

Android Development

Xtend works great on Android, as it doesn't produce additional runtime overhead. The very thin lib and the advanced support for code generation are increasing productivity while helping to keep your Android apps small.

Web Development

The Google Web Toolkit translates Java source code to fast Javascript code. Xtend makes typical GWT programming a joy. There are many nice examples and cool enhancements out there.

Xtend is a statically-typed programming language which translates to comprehensible Java source code. Syntactically and semantically Xtend has its roots in the Java programming language but improves on many aspects:

- **Extension methods** - enhance closed types with new functionality
- **Lambda Expressions** - concise syntax for anonymous function literals
- **ActiveAnnotations** - annotation processing on steroids
- **Operator overloading** - make your libraries even more expressive
- **Powerful switch expressions** - type based switching with implicit casts
- **Multiple dispatch** - a.k.a. polymorphic method invocation
- **Template expressions** - with intelligent white space handling
- **No statements** - everything is an expression
- **Properties** - shorthands for accessing and defining getters and setter
- **Type inference** - you rarely need to write down type signatures anymore
- **Full support for Java generics** - including all conformance and conversion rules
- **Translates to Java** not bytecode - understand what is going on and use your code for platforms such as Android or GWT

Features (1)

- Extension methods - enhance closed types with new functionality
- Lambda Expressions - concise syntax for anonymous function literals
- ActiveAnnotations - annotation processing on steroids
- Operator overloading - make your libraries even more expressive
- Powerful switch expressions - type based switching with implicit casts
- Multiple dispatch - a.k.a. polymorphic method invocation

Features (2)

- Template expressions - with intelligent white space handling
- No statements - everything is an expression
- Properties - shorthands for accessing and defining getters and setter
- Type inference - you rarely need to write down type signatures anymore
- Full support for Java generics - including all conformance and conversion rules
- Translates to Java not bytecode - understand what is going on and use your code for platforms such as Android or GWT

Hello World

xtend

```
class HelloWorld
{
  def static void main(String[] args)
  {
    println("Hello World")
  }
}
```

java

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello World");
  }
}
```

Selected Key Features

- Java Interoperability
  - Type Inference
  - Conversion Rules
- Classes
  - Constructors
  - Fields
  - Methods
  - Override
  - Inferred return types

- Expressions
  - Literals
  - Casts
  - Field access & method invocation
  - Constructor call
  - Lambda Expressions
  - If expression
  - switch Expression
  - return expression

- Annotations
  - @Data

Java Interoperability

Type inference

- Xtend, like Java, is a statically typed language.
- It completely supports Java's type system, including the primitive types like int or boolean, arrays and all the Java classes, interfaces, enums and annotations that reside on the class path.
- With Java is that you are forced to write type signatures over and over again - one of the reasons people do not like static typing.
- It is not a problem of static typing but simply a problem with Java.
- Although Xtend is statically typed just like Java, you rarely have to write types down because they can be computed from the context.

```java
public static void main(String[] args)
{
  List<String> names = new ArrayList<String>();
  names.add("Ted");
  names.add("Fred");
  names.add("Jed");
  names.add("Ned");
  System.out.println(names);
  Erase e = new Erase();
  List<String> short_names = e.filterLongerThan(names, 3);
  System.out.println(short_names.size());
  for (String s : short_names)
  {
    System.out.println(s);
  }
}
```

- Java

```
def static void main(String[] args)
{
  var names = new ArrayList<String>()
  names.add("Ted")
  names.add("Fred")
  names.add("Jed")
  names.add("Ned")
  System.out.println(names)
  var e = new Erase()
  var  short_names = e.filterLongerThan(names, 3)
  System.out.println(short_names.size())
  for (s : short_names)
  {
    System.out.println(s)
  }
}
```

- XTend

```
public static void main(String[] args)
{
  List<String> names = new ArrayList<String>();
  names.add("Ted");
  names.add("Fred");
  names.add("Jed");
  names.add("Ned");
  System.out.println(names);
  Erase e = new Erase();
  List<String> short_names = e.filterLongerThan(names, 3);
  System.out.println(short_names.size());
  for (String s : short_names)
  {
    System.out.println(s);
  }
}
```

- Java

```
def static void main(String[] args)
{
  var names = new ArrayList<String>()
  names.add("Ted")
  names.add("Fred")
  names.add("Jed")
  names.add("Ned")
  System.out.println(names)
  var e = new Erase()
  var  short_names = e.filterLongerThan(names, 3)
  System.out.println(short_names.size())
  for (s : short_names)
  {
    System.out.println(s)
  }
}
```

- XTend

Conversion Rules

- In addition to Java's autoboxing to convert primitives to their corresponding wrapper types (e.g. int is automatically converted to Integer when needed), there are additional conversion rules in Xtend.
- Arrays are automatically converted to List<ComponentType> and vice versa.
- Subsequent changes to the array are reflected by the list and vice versa.
- Arrays of primitive types are converted to lists of their respective wrapper types.
- All subtypes of Iterable are automatically converted to arrays on demand

```
def toList(String[] array)
{
  val List<String> asList = array
  return asList
}
```

Classes

Classes

- At a first glance an Xtend file pretty much looks like a Java file.
- It starts with a package declaration followed by an import section and class definitions.
- The classes in fact are directly translated to Java classes in the corresponding Java package.
- A class can have constructors, fields, methods and annotations.

```
package acme.com

import java.util.List

class MyClass
{
  String name

  new(String name)
  {
    this.name = name
  }

  def String first(List<String> elements)
  {
    elements.get(0)
  }
}
```

Class Declaration

- The class declaration reuses a lot of Java's syntax but still is a bit different in some aspects:
  - All Xtend types are public by default since that's the common case.
  - Java's "package private" default visibility is declared by the more explicit keyword package in Xtend.
  - Xtend supports multiple public top level class declarations per file. Each Xtend class is compiled to a separate top-level Java class.
  - Abstract classes are defined using the abstract modifier as in Java

Constructors

- An Xtend class can define any number of constructors.
- Unlike Java you do not have to repeat the name of the class over and over again, but use the keyword new to declare a constructor.
- Constructors can also delegate to other constructors using this(args...) in their first line

```
class MyClass
{
  String name

  new(String name)
  {
    this.name = name
  }

  def String first(List<String> elements)
  {
    elements.get(0)
  }
}

class MySpecialClass extends MyClass
{
  new(String s)
  {
    super(s)
  }

  new()
  {
    this("default")
  }
}
```

Fields

- A field can have an initializer.
- Final fields are declared using val, while var introduces a non-final field and can be omitted.
- If an initializer expression is present, the type of a field can be inferred if val or var was used to introduce the field.
- The keyword final is synonym to val.
- Fields marked as static will be compiled to static Java fields.
- The default visibility for fields is private. You can also declare it explicitly as being public, protected,package or private.

```
class MyDemoClass
{
  int count = 1
  static boolean debug = false
  var name = 'Foo' // type String is inferred
  val UNIVERSAL_ANSWER = 42 // final field with inferred type int
}
```

Methods

- Xtend methods are declared within a class and are translated to a corresponding Java method with exactly the same signature.
- Method declarations start with the keyword def.
- The default visibility of a method is public.
- You can explicitly declare it as being public, protected, package or private.
- It is possible to infer the return type of a method from its body.

```
class MyClass
{
  String name

  new(String name)
  {
    this.name = name
  }

  def String first(List<String> elements)
  {
    elements.get(0)
  }

  def static createInstance()
  {
    new MyClass('foo')
  }
}
```

Overriding Methods

- Methods can override other methods from the super class or implement interface methods using the keyword override.
- If a method overrides a method from a super type, the override keyword is mandatory and replaces the keyword def.
- The override semantics are the same as in Java, e.g. it is impossible to override final methods or invisible methods.
- Overriding methods inherit their return type from the super declaration.

```
class MyClass
{
  String name

  new(String name)
  {
    this.name = name
  }

  def String first(List<String> elements)
  {
    elements.get(0)
  }
}

class MySpecial extends MyClass
{
  new(String s)
  {
    super(s)
  }

  override first(List<String> elements)
  {
    elements.get(1)
  }
}
```

Inferred Return Types

- If the return type of a method can be inferred from its body it does not have to be declared.

```
def String first(List<String> elements)
{
  elements.get(0)
}
```

```
def first(List<String> elements)
{
  elements.get(0)
}
```

Expressions

Literals

- A literal denotes a fixed, unchangeable value.
- Literals for
  - strings,
  - numbers,
  - booleans,
  - null
- and Java types are supported as well as literals for collection types like
  - lists,
  - sets
  - maps
  - arrays.

```
42
1_234_567_890
0xbeef    // hexadecimal
077       // decimal 77 (*NOT* octal)
-1  // an expression consisting of
    // the unary - operator and an integer literal
42L
0xbeef#L // hexadecimal, mind the '#'
0xbeef_beef_beef_beef_beef#BI // BigInteger
42d     // double
0.42e2  // implicit double
0.42e2f // float
4.2f    // float
```

Collection Literals

- Convenient to create instances of the various collection types the JDK offers.

```
val myList = newArrayList('Hello', 'World')
val myMap = newLinkedHashMap('a' -> 1, 'b' -> 2)
```

- Xtend supports collection literals to create immutable collections and arrays, depending on the target type.

```
val myList = #['Hello','World']
```

- If the target type is an array, an array is created instead without any conversion:

```
val String[] myArray = #['Hello','World']
```

- An immutable set can be created using curly braces instead of the squared brackets:

```
val mySet = #{'Hello','World'}
```

- An immutable map is created like this:

```
val myMap = #{'a' -> 1 ,'b' ->2}
```

Type Casts

- A type cast behaves exactly like casts in Java, but has a slightly more readable syntax.

- Type casts bind stronger than any other operator but weaker than feature calls

```
something as MyClass

42 as Integer
```

Null safe Feature Call

- Checking for null references can make code very unreadable.

```
if (myRef != null) myRef.doStuff()
```

- In many situations it is ok for an expression to return null if a receiver was null.

- Xtend supports the safe navigation operator ?. to make such code better readable.

```
myRef?.doStuff
```

Elvis Operator

- In addition to null-safe feature calls Xtend supports the elvis operator known from Groovy.
- The right hand side of the expression is only evaluated if the left side was null.

```
val salutation = person.firstName ?: 'Sir/Madam'
```

Variable Declarations

- A variable declaration starting with the keyword val denotes a value, which is essentially a final, unsettable variable.

- The variable needs to be declared with the keyword var, which stands for 'variable' if it should be allowed to reassign its value.

```
{
  val max = 100
  var i = 0
  while (i < max)
  {
    println("Hi there!")
    i = i + 1
  }
}
```

Typing

- The type of the variable itself can either be explicitly declared or it can be inferred from the initializer expression.

```
var List<String> strings = new ArrayList
```

- In such cases, the type of the right hand expression must conform to the type of the expression on the left side.

- Alternatively the type can be inferred from the initializater

```
var strings = new ArrayList<String>
```

Constructor Call

- Constructor calls have the same syntax as in Java.
- The only difference is that empty parentheses are optional:

```
new String() == new String
new ArrayList<BigDecimal>() == new ArrayList<BigDecimal>
```

- If type arguments are omitted, they will be inferred from the current context similar to Java's diamond operator on generic method and constructor calls.

Lambda Expressions (1)

- A lambda expression is basically a piece of code, which is wrapped in an object to pass it around.
- As a Java developer it is best to think of a lambda expression as an anonymous class with a single method
- This kind of anonymous classes can be found everywhere in Java code and have always been the poor-man's replacement for lambda expressions in Java.

```java
// Java Code!
final JTextField textField = new JTextField();
textField.addActionListener(new ActionListener()
{
  @Override
  public void actionPerformed(ActionEvent e)
  {
    textField.setText("Something happened!");
  }
});
```

Lambda Expressions (2)

- Xtend not only supports lambda expressions, but offers an extremely dense syntax for it.

```
// Java Code!
final JTextField textField = new JTextField();

textField.addActionListener(new ActionListener()
{
  @Override
  public void actionPerformed(ActionEvent e)
  {
    textField.setText("Something happened!");
  }
});
```

```
val textField = new JTextField

textField.addActionListener([ ActionEvent e |
  textField.text = "Something happened!"
])
```

Lambda Expressions (3)

- lambda expression is surrounded by square brackets (inspired from Smalltalk).
- Also a lambda expression like a method declares parameters.
- The lambda here has one parameter called e which is of type ActionEvent.
- You do not have to specify the type explicitly because it can be inferred from the context

```
textField.addActionListener([ e |
  textField.text = "Something happened!"
])
```

Lambda Expressions (4)

- Also as lambdas with one parameter are a common case, there is a special short hand notation for them, which is to leave the declaration including the vertical bar out.

```
textField.addActionListener([
  textField.text = "Something happened!"
])
```

- The name of the single variable will be **it** in that case.

- Since you can leave out empty parentheses for methods which get a lambda as their only argument, you can reduce the code above further down.

```
textField.addActionListener [textField.text = "Something happened!"]
textField.addActionListener(new ActionListener()
{
  @Override
  public void actionPerformed(ActionEvent e)
  {
    textField.setText("Something happened!");
  }
});
```

```
textField.addActionListener([ ActionEvent e |
  textField.text = "Something happened!"
])
```

```
textField.addActionListener([ e |
  textField.text = "Something happened!"
])
```

```
textField.addActionListener([
  textField.text = "Something happened!"
])
```

```
textField.addActionListener [textField.text = "Something happened!"]
```

Lambdas & Collections

- The collections have been equipped with Extension Methods that take lambda as parameters

- *Can dramatically reduce number of loops in a program!*

```
def printAll(String... strings)
{
  strings.forEach[ s | println(s) ]
}
```

```
list.forEach[ element, index |
 .. // if you need access to the current index
]
list.reverseView.forEach[
  .. // if you just need the element it in reverse order
]
```

```
val strings = list("red", "blue", "green")
val charCount = strings.map[s|s.length].reduce[sum, size | sum + size]
```

Switch Expression

- The switch expression is very different from Java's switch statement:
    - there is no fall through which means only one case is evaluated at most.
    - The use of switch is not limited to certain values but can be used for any object reference.
    - Object.equals(Object) is used to compare the value in the case with the one you are switching over.

```
switch myString
{
  case myString.length > 5 : "a long string."
  case 'some' : "It's some string."
  default : "It's another short string."
}
```

Switch Expression- Type guards

- Instead of or in addition to the case guard you can specify a type guard.
- The case only matches if the switch value conforms to this type.
- A case with both a type guard and a predicate only matches if both conditions match.
- If the switch value is a field, parameter or variable, it is automatically casted to the given type within the predicate and the case body.

```
def length(Object x)
{
  switch x
  {
    String case x.length > 0 : x.length
                    // length is defined for String
    List<?> : x.size
                    // size is defined for List
    default : -1
  }
}
```

Active Annotations

Active Annotations

- Xtend comes with ready-to-use active annotations for common code patterns.
- They reside in the org.eclipse.xtend.lib plug-in/jar which must be on the class path of the project containing the Xtend files.

@Data

- The annotation @Data will turn an annotated class into a value object class. A class annotated with @Data is processed according to the following rules:
  - all fields are final,
  - getter methods will be generated (if they do not yet exist),
  - a constructor with parameters for all non-initialized fields will be generated (if it does not exist),
  - equals(Object) / hashCode() methods will be generated (if they do not exist),
  - a toString() method will be generated (if it does not exist).

```
@Data class Person
{
  String firstName
  String lastName
}
```

```
@Data class Person
{
  String firstName
  String lastName
}
```

```
@Data
@SuppressWarnings("all")
public class Person {
  private final String _firstName;
  public String getFirstName() {
    return this._firstName;
  }
  private final String _lastName;
  public String getLastName() {
    return this._lastName;
  }
  public Person(final String firstName, final String lastName) {
    super();
    this._firstName = firstName;
    this._lastName = lastName;
  }
  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((_firstName== null) ? 0 : _firstName.hashCode());
    result = prime * result + ((_lastName== null) ? 0 : _lastName.hashCode());
    return result;
  }
  @Override
  public boolean equals(final Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    Person other = (Person) obj;
    if (_firstName == null) {
      if (other._firstName != null)
        return false;
    } else if (!_firstName.equals(other._firstName))
      return false;
    if (_lastName == null) {
      if (other._lastName != null)
        return false;
    } else if (!_lastName.equals(other._lastName))
      return false;
    return true;
  }
  @Override
  public String toString() {
    String result = new ToStringHelper().toString(this);
    return result;
  }
}
```

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see http://creativecommons.org/licenses/by-nc/3.0/

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit