Using the Eclipse IDE for Java programming - Tutorial

Lars Vogel, (c) 2007 - 2020 vogella GmbH – Version 5.4, 29.09.2020

1. The Eclipse IDE for Java development

The Eclipse IDE (integrated development environment) provides strong support for Java developer. In 2020 Eclipse is one of the leading IDEs with approximately one millions downloads per month.

Eclipse can be extended with additional software components called plug-ins. Pre-packaged Eclipse distributions provide a consistent set of functionality.

Eclipse IDE downloads provides Eclipse distributions for different use cases.

The Eclipse IDE for Java Developers distribution is designed to support standard Java development. It includes support for the Maven and Gradle build system and support for the Git version control system. == Install the Eclipse Java IDE

TABLE OF CONTENTS

You can install the Eclipse Java IDE via an installer or via a packaged download.

1.1. Using the installer

Download the installer via `https://eclipse.org/downloads/eclipse-packages/`.

On Linux you have to unzip it before you can start it.On Mac the installer is delivered as packaged application and can be installed and started regular Mac installation procedures.On Windows and Mac you can run it directly via the delivered executable / package application.

Pick Eclipse IDE for Java Developers from the list and perform the installation.

You may have to confirm some license agreements.

After the installation you can start the IDE directly. Remember the installation folder, in this folder you find the Eclipse installation to start it again.

1.2. Using a packaged download

In case you don't want to use the installer, you can also directly download the Eclipse IDE for Java Developers package from: `https://eclipse.org/downloads/eclipse-packages/`

Press on the link beside the package description, for example Linux 64-Bit to start the download.The links which are displayed depend on your operating system.

The download depend on your platform, it is either a compressed archive of multiple files or a packaged application (Mac).

After you downloaded the file with the Eclipse distribution, unpack it to a local directory or follow the installation procedure on Mac.

> As a developer person you probably know how to extract a compressed file or how to install apps on Mac.
>
> But if in doubt, google for "How to extract a zip on Window" or "How to unzip a file on Linux" or "How to install a dmg file on Mac"

2. Starting the IDE and picking a workspace

To start Eclipse, double-click the `eclipse.exe` (Microsoft Windows) or `eclipse` (Linux / Mac) file from your installation directory.

> *What to do if the Eclipse IDE does not start*
>
> The Eclipse IDE requires at least Java 11 to run.If Eclipse does not start, check your Java version.

The Eclipse IDE prompts you for a workspace to store it configuration.Select an empty directory and click the `OK` button.

Eclipse starts and shows the Welcome page.Close this page by clicking the x beside Welcome.

After closing the welcome screen, the application should look similar to the following screenshot.

2.1. Appearance

By default, Eclipse ships in a light configuration, if you prefer you can switch to a dark theme via **Window □Preferences □ General □Appearance** menu.The Theme selection allows you to switch to the Dark theme of Eclipse.

Restart your IDE afterwards, some native OS styling functionality requires a restart.

3. Important Eclipse terminology

3.1. Workspace

The workspace is the physical location (file path) for storing meta-data and (optional) your development artifacts.The meta-data stored for the workspace contains preferences settings, plug-in specific meta data, logs etc.

You can choose the workspace during startup of Eclipse or via the **File □Switch Workspace □Others** menu entry.

Your projects, source files, images and other artifacts can be stored inside or outside your workspace.For example, if you use Git as version control system, you typically would store the Git repositories outside of the workspace.

3.2. Overview of the user interface

Eclipse provides views and editors to navigate and change content. View and editors can be grouped into perspectives.

More on perspectives

A view is typically used to display structured data and allow to modify it directly.

For example, the Project Explorer view allows you to browse and modify files of Eclipse projects. If you rename a file via the Project Explorer the file name is directly changed without having to save.

Editors are typically used to modify a single data element, for example a text file. To apply these changes to the underlying data mode, you need to select save from the menu or the toolbar. A editor with unsaved data (a dirty editor) is marked with an asterisk left to the name of the modified file.

3.3. Eclipse projects

An Eclipse project contains source, configuration and binary files related to a certain task. It groups them into buildable and reusable units. An Eclipse project can have natures assigned to it which describe the purpose of this project. For example, the Java nature defines a project as Java project. Projects can have multiple natures combined to model different technical aspects.

Natures for a project are defined via the .project file in the project directory.

4. The Eclipse Java perspective

The following description is a small introduction into important elements of the Java perspective. Feel free to skip this chapter as it only can be used as a reference.

4.1. Package Explorer view

The Package Explorer view allows you to browse the structure of your projects and to open files in an editor via a double-click on the file.

It is also used to change the structure of your project. For example, you can rename files or move files and folders via drag and drop. A right-click on a file or folder shows you the available options.

Package Explorer

For more info on the Package Explorer see Chapter Source Navigation and Link Package Explorer with editor.

4.2. Outline view

The Outline view shows the structure of the currently selected source file.

4.3. Problems view

The Problems view shows errors and warning messages. Sooner or later you will run into problems with your code or your project setup. To view the problems in your project, you can use the Problems view which is part of the standard Java perspective. If this view is closed, you can open it via **Window → Show View → Problems**.

The messages which are displayed in the Problems view can be configured via the drop-down menu of the view. For example, to display the problems from the currently selected project, select Configure Contents and set the Scope to On any element in the same project.

The Problems view also allows you to trigger a Quick fix via a right mouse-click on several selected messages. See chapter Quick Fix for details on the Quick fix functionality.

4.4. Javadoc view

The Javadoc view shows the documentation of the selected element in the Java editor.

4.5. Java editor

The Java editor is used to modify the Java source code. Each Java source file is opened in a separate editor.

If you click in the left column of the editor, you can configure its properties, for example, that line number should be displayed.

5. Create your first Java program

The following section describes how to create a minimal Java application using the Eclipse IDE.

5.1. Create project

Select **File □New □Java project** from the menu.Enter `com.vogella.eclipse.ide.first` as the project name and press the `Finish` button to create the project.

A new project is created and displayed as a folder.Open the `com.vogella.eclipse.ide.first` folder and explore the content of this folder.

> In this tutorial the project is typically named the same as the top-level Java package in the project.This makes is easier to find a project related to a piece of code.

5.2. Create package

> A good naming convention is to use the same name for the top level package and the project.For example, if you name your project `com.example.javaproject` you should also use `com.example.javaproject` as the top-level package name.

Create the `com.vogella.eclipse.ide.first` package by seleting the `src` folder, right-click on it and select **New** □

Package.

Press the `Finish` button.

5.3. Create Java class

Right-click on your package and select **New □Class** to create a Java class.

Enter `MyFirstClass` as the class name and select the public static void main (String[] args) checkbox.

Press the `Finish` button.

This creates a new file and opens the Java editor. Change the class based on the following listing.

```java
package com.vogella.eclipse.ide.first;

public class MyFirstClass {

  public static void main(String[] args) {
    System.out.println("Hello Eclipse!");
  }

}
```

You could also directly create new packages via this dialog. If you enter a new package in this dialog, it is created automatically.

5.4. Run your application code from the IDE

Now run your code. Either right-click on your Java class in the Package Explorer or right-click in the Java class and select **Run-as □Java application**.

Eclipse will run your Java program. You should see the output in the Console view.

Congratulations! You created your first Java project, a package, a Java class and you ran this program inside Eclipse.

6. Run Java program outside Eclipse

6.1. Create JAR file

To run the Java program outside of the Eclipse IDE, you need to export it as a JAR file. A JAR file is the standard distribution format for Java applications.

Select your project, right-click it and select the Export menu entry.

Select JAR file and select the `Next` button. Select your project and enter the export destination and a name for the JAR file, for example `myprogram.jar`.

Press The `Finish` button. This creates a JAR file in your selected output directory.

6.2. Run your program outside Eclipse

Open a command shell, e.g., under Microsoft Windows select **Start □Run** and type `cmd` and press the Enter key. This should open a console window.

Switch to the directory which contains the JAR file, by typing `cd path`. For example, if your JAR is located in c:\temp, use the following command.

```
cd c:\temp
```

To run this program, include the JAR file in your classpath. The classpath defines which Java classes are available to the Java runtime. You can add a JAR file to the classpath with the -classpath option.

```
java -classpath myprogram.jar
de.vogella.eclipse.ide.first.MyFirstClass
```

Type the above command in the directory you used for the export and you see the "Hello Eclipse!" output in your command shell.

7. Exercise: Java project, packages and import statements

7.1. Create project

Create a new Java project called com.vogella.ide.counter.

Create the following packages:

- `com.vogella.ide.counter.util`
- `com.vogella.ide.counter.main`

7.2. Create classes

Create the following `Counter` class in the `*.util` package.

```
package com.vogella.ide.counter.util;
```

```
public class Counter {
  public int count (int x){
    // TODO check that x > 0 and <= 255
    // if not throw a new RuntimeException
    // Example for a RuntimeException:

    // throw new RuntimeException("x should be between 1
and 255");

    // TODO calculate the numbers from 1 to x
    // for example if x is 5, calculate
    // 1 + 2 + 3 + 4 + 5


    // TODO return your calculated value
    // instead of 0
    return 0;
  }
}
```

Create the following `Tester` class in the `*.main` package. This is a simple class without the usage of any unit testing framework like JUnit.The Eclipse editor should mark the created class with an error because the required `import` statements are missing.

```
package com.vogella.ide.counter.main;

public class Tester {

  public static void main(String[] args) {
    Counter counter = new Counter();
  }


}
```

Right-click in your Java editor and select **Source Organize Imports** to add the required import statements to your Java class.

This should remove the syntax error. Finish the implementation for the `Tester` class based on the following code.

```java
package com.vogella.ide.counter.main;

import com.vogella.ide.counter.util.Counter;

public class Tester {

  public static void main(String[] args) {
    Counter counter = new Counter();
    int result = counter.count(5);
    if (result == 15) {
      System.out.println("Correct");
    } else {
      System.out.println("False");
    }
    try {
      counter.count(256);
    } catch (RuntimeException e) {
      System.out.println("Works as exepected");
    }
  }

}
```

The `Counter` class had in its source code a comment starting with TODO.Finish the source code and calculate the correct values.

Run the `Tester` class and validate that your implementation is correct.The `Tester` class checks for an example value but the method should work for different input values.

8. Exporting and importing projects

8.1. Exporting projects

You can export and import Eclipse projects.□This allows you to share projects with other people and to import existing projects.

To export Eclipse projects, select **File** □**Export** □**General** □ **Archive File** and select the projects you want to export.

8.2. Importing projects

To import projects, select□**File** □**Import** □**Existing Projects into Workspace**. You can import from an archive file, i.e., zip file or directly□import the projects in case you have extracted the zip file.

8.3. Exercise: Export and import projects

Export your one of your projects into a zip file.□Switch into a new workspace and import the project into your new workspace based on the zip file you exported.

9. Source navigation in the Eclipse IDE

9.1. Package Explorer or Project Explorer

The primary way of navigating through your project is the Package Explorer or alternatively the Project Explorer view. You can open nodes in the tree and open a file in an editor by double-clicking on the corresponding entry in the tree hierarchy.

The drop-down menu in the Package Explorer allows you to filter the resources which should be displayed or hidden.

9.2. Link Package Explorer with editor

The Package Explorer view allows you to display the associated file from the currently selected editor. For example, if you are working on the `Foo.java` file in the Java editor and switch to the Java editor of the `Var.java` file, then the corresponding file will be selected in the Package Explorer view.

To activate this behavior, press the `Link with Editor` button in the Package explorer view as depicted in the following screenshot.

9.3. Opening a class

You can navigate between the classes in your project via the Package Explorer view as described before. You can navigate the tree and open a file via a double-click.

In addition, you can open any class by positioning the cursor on the class in an editor and pressing `F3`. Alternatively, you can press `Ctrl`+`Shift`+`T`. This shows the following dialog in which you can enter the class name to open it.

You can also search for package names. Each part of the package name must end with a `.` (the dot character) so that the Open Type Dialog can identify it as a package.

You only need to specify part of each segment of the package name. Assume, for example, that you search for the org.eclipse.swt.widgets.Button class. To find this class, you can use the search term org.eclipse.swt.widgets.Button or o.e.s.w.Button or o.Button.

The Open Type Dialog also supports CamelCase like search, e.g., it matches capital letters in the class name. For example, if you would search for the `OnTouchListener` class you could use OTL or OToList as search term.

To avoid suffix matching, you can add a space after the class name. For example, you can type Selection (there is a space after selection) to match the `Selection` class but not the `SelectionListener` class. Wildcards like * are also supported.

9.4. Open Resource dialog to open arbitrary files

You can open any file from your open projects via the Open Resource dialog. You can open this dialog via the `Ctrl`+`Shift`+`R` shortcut. This dialog allows to enter the file name and to open or show it in a selected view. The following screenshot demonstrate the usage to open a pom.xml file from your workspace.

9.5. Quick Outline

Quick Outline shows you an structured overview of the file you are editing. For example, for a Java class you see its methods with the option to filter. The shortcut for opening the Quick Outline is `Ctrl`+`O`. You can also reach this option, via right-click in an editor via the Quick Outline option.

By default, Quick Outline shows only the direct members and fields of the Java class. Press `Ctrl`+`O` again to show also the inherited members and fields.

The default look of the Quick Outline option is similar to the Quick Outline view of the Java perspective.

9.6. Open Type Hierarchy

The type hierarchy of a class shows you which classes it extends and which interfaces it implements. You can use the type hierarchy to navigate to one of these elements.

To open the type hierarchy of the selected class, right-click in the editor and select Open Type Hierarchy (Shortcut: `F4`) or Quick Type Hierarchy (Shortcut: `Ctrl`+`T`).

9.7. Full text search

You frequently need to find files containing certain text or other meta data. Via the **Search** □Search... (Shortcut: `Ctrl`+`H`) you can open the search dialog of Eclipse. Use the File Search tab to search for text with the option to use regular expressions and also to replace matching entries.

Eclipse associates file extensions with the default tab. You can customize the available search tabs via the `Customize` button in the Search dialog. Via the Remember the last used page you can configure Eclipse to use your last tab as default.

9.8. Java search and other specialized searches

The Search functionality (Ctrl+H) offers specialized searches for more complex use cases. For example, use the Java Search tab to search for Java elements, e.g., methods.

The Search view shows the search results for the selected scope. You can double-click on a search entry to navigate to the corresponding position in the editor. The currently selected search result is also indicated via an arrow in the left border of the editor.

9.9. Inline search in an editor

You can use the Ctrl+J shortcut to activate Incremental Find. This allows you to search in the current active editor for a text which is displayed in the status line as depicted by the following screenshot. Repeat Ctrl+J in order to move to the next occurrence of the current search term.

The advantage of this search is that no pop-up dialog is opened which blocks other elements in the Eclipse IDE.

If you have selected an element in the editor, you can use the Ctrl+K shortcut to search for the next occurrence of the selected text and Ctrl+Shift+K for the previous element.

9.10. Annotation navigations

You can also navigate via the annotation buttons, e.g., for jumping to the next error or warning in your source code.

By pressing the buttons you can navigate to the related annotations. You can also use the keyboard shortcut `Ctrl`+`.` (Ctrl plus the dot sign) for selecting the next annotation or `Ctrl`+`,` (Ctrl plus the comma sign) for selecting the previous annotation.

The following screenshot shows source code with two warnings and one error and you can navigate between the corresponding code via the annotation buttons.

Which annotations are relevant for navigation can be configured via the drop-down menu of the toolbar. This selection is highlighted in the following screenshot.

9.11. Mouse and keyboard navigation

In a lot of cases you can also use the mouse to navigate to or into an element if you press the `Ctrl` key. For example, press the `Ctrl` key and (left) click with the mouse on the name of a class to jump into the class declaration.

Similar to the left mouse click combined with the `Ctrl`, you can use the `F3` key to go into a class.

9.12. Show in Breadcrumb

You can also activate the breadcrumb mode for the Java editor which allows you to navigate the source code directly from the Java editor.

You can activate this mode via right-click in the editor and by selecting the Show in Breadcrumb entry.

This allows you to navigate the source code from the editor as depicted in the following screenshot.

To hide it again, right-click on a breadcrump entry and select Hide Breadcrumb.

9.13. Shortcuts

There are a lot of shortcuts available for navigation. Please check the appendix of this tutorial for these shortcuts or open **Window+Preferences** **General** **Keys** to find and redefine shortcuts at runtime.

9.14. Closing and opening projects

Closing projects saves memory in Eclipse and can reduce the build time. Eclipse ignores closed projects, e.g., all searches ignore files from closed projects. Also the Problems view does only shows errors of opened projects. This typically helps you focus your attention on the project. You can close projects via a right-click on it and by selecting the Close Project menu entry. Alternatively, if you work on a project, you can close all unrelated projects via a right-click on it and by selecting the Close Unrelated Projects menu entry.

To open a closed project double-click on it, or right-click it and select Open Project.

You can use the filter functionality for the Package Explorer view to hide the closed projects.

10. Content Assist and Quick Fix

10.1. Content assist

Content assist is a functionality in Eclipse which allows the developer to get context-sensitive code completion in an editor upon user request.

It can be invoked by pressing `Ctrl`+`Space`.

For example, type `syso` in the editor of a Java source file and then press `Ctrl`+`Space`. This will replace `syso` with `System.out.println("")`.

If you have a reference to an object, for example, the object `person` of the type `Person` and need to see its methods, type `person.` and press `Ctrl`+`Space`.

10.2. Quick Fix

Whenever Eclipse detects a problem, it will underline the problematic text in the editor. Select the underlined text and press `Ctrl`+`1` to see proposals how to solve this problem. This functionality is called Quick Fix.

For example, type `myBoolean = true;` If myBoolean is not yet defined, Eclipse will highlight it as an error. Select the variable and press `Ctrl`+`1`. Eclipse will suggest creating a field or local variable.

Quick Fix is extremely powerful. For example, it allows you to create new local variables and fields as well as new methods and new classes. Or it can put `try/catch` statements around your exceptions. It can also assign a statement to a variable and much more.

Quick Fix also gives several options for code changes on code which does not contain errors, e.g., it allows you to convert a local variable to a field.

10.3. Exercise: Convert anonymous inner classes to lambda expressions and vice versa

The Eclipse IDE has full support for modern Java versions. This section demonstrates the quick fix for converting anonymous inner classes to lambda expressions.

```java
package com.vogella.eclipse.ide.java8;

public class Java8Example {

    public static void main(String[] args) {
        Runnable runnable = new Runnable() {
            @Override
            public void run() {
                System.out.println("Hello Lambdas");
            }
        };

        new Thread(runnable);
    }
}
```

You can use a quick fix for the conversion as demonstrated via the following screenshots.

11. Generating code

Eclipse has several possibilities to generate code for you. This can save significant time during development.

For example, Eclipse can override methods from superclasses and generate the `toString()`, `hashcode()` and `equals()` methods. It can also generate getter and setter methods for attributes of your Java class.

You can find these options in the Source menu.

To test the source generation, create the following class in your `com.vogella.eclipse.ide.first` project.

```java
package com.vogella.eclipse.ide.first;

public class Person {
  private String firstName;
  private String lastName;

}
```

Select **Source** Generate Constructor using Fields..., mark both fields and click the `OK` button.

Select **Source** Generate Getter and Setter..., select all fields and afterwards click the `OK` button.

Select **Source** Generate toString()..., mark again all fields and click the `OK` button.

You created the following class:

```java
package com.vogella.eclipse.ide.first;

public class Person {
  private String firstName;
  private String lastName;

  public Person(String firstName, String lastName) {
    super();
    this.firstName = firstName;
    this.lastName = lastName;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  @Override
  public String toString() {
    return "Person [firstName=" + firstName + ", lastName="
+ lastName
        + "]";
  }

}
```

12. Exercise: code generation and content assists

12.1. Introduction

In this exercise you practice the usage of code generation and the usage of the Content Assists functionality.

12.2. Create project

Create a project called `com.vogella.ide.todo`.

12.3. Create class

Create the `com.vogella.ide.todo` package and the following class.

```
package com.vogella.ide.todo;

import java.util.Date;

public class Todo {

  private long id;
  private String summary = "";
  private String description = "";
  private boolean done = false;
  private Date dueDate;

}
```

Select **Source** Generate Constructor using Fields... to generate a constructor using all fields.

Use the **Source** **Generate Getter and Setter** to create getters and setters for all fields.

The resulting class should look like the following listing.

```
package com.vogella.ide.todo;

import java.util.Date;

public class Todo {

  private long id;
  private String summary = "";
  private String description = "";
  private boolean done = false;
  private Date dueDate = new Date();

  public Todo(long id, String summary, String description,
boolean done, Date dueDate) {
     this.id = id;
     this.summary = summary;
     this.description = description;
     this.done = done;
```

```
        setDueDate(dueDate);
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getSummary() {
        return summary;
    }

    public void setSummary(String summary) {
        this.summary = summary;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean isDone() {
        return done;
    }

    public void setDone(boolean done) {
        this.done = done;
    }

    public Date getDueDate() {
        return dueDate;
    }

    public void setDueDate(Date dueDate) {
            this.dueDate = new Date(dueDate.getTime());
        }

}
```

Use Eclipse to generate a `toString()` method for the `Todo` class based on the id and summary field.

This can be done via the Eclipse menu **Source** Generate toString()....

Also generate a `hashCode()` and `equals()` method based on the id field. This can be done via the **Source** Generate hashCode() and equals()... menu entry.

12.4. Create instances

Create a new class called `TodoProvider`. Create the following static method in your `TodoProvider` class.

```
package com.vogella.ide.todo;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class TodoProvider {
    private static int current = 0;

    // example data, change if you like
    public static List<Todo> createInitialModel() {
        ArrayList<Todo> list = new ArrayList<Todo>();
        list.add(createTodo("SWT", "Learn Widgets"));
        list.add(createTodo("JFace", "Especially
Viewers!"));
        list.add(createTodo("DI", "@Inject looks
interesting"));
        list.add(createTodo("OSGi", "Services"));
        list.add(createTodo("Compatibility Layer", "Run
Eclipse 3.x"));
        return list;
    }

    private static Todo createTodo(String summary, String
description) {
        return new Todo(current++, summary, description,
false, new Date());
    }
}
```

12.5. Write a test class

Write another `TodoProviderTest` class with a `public static void main (String[] args)` method.

In your main method call the `createInitialModel` method and validate that the returned number of items is 5.

If another number than 5 is returned, throw a `RuntimeException`.If the correct number is returned, write the String "Correct" to the Console view.

Use Content assist to create the `System.out.println()` based on `syso` for you.

12.6. Example implementation of TodoProviderTest

While this exercise was about code generation and content assists, you might be interested in a potential solution for this exercise. The following listing contains a potential solution.

```
package com.vogella.ide.todo;

import java.util.List;

public class TodoProviderTest {

    public static void main(String[] args) {
        List<Todo> model =
TodoProvider.createInitialModel();
        if (model.size()!=5){
            throw new RuntimeException("size should be 5");
        } else {
            System.out.println("Correct");
        }
    }

}
```

13. Refactoring

This section covers the refactoring facilities of Eclipse which allow you to improve the structure of your source code.

13.1. Refactoring

Refactoring is the process of restructuring the code without changing its behavior. For example, renaming a Java class or method is a refactoring activity.

13.2. Refactoring in Eclipse

Eclipse supports several refactoring activities, for example, renaming or moving.

For example, to use the Rename refactoring, you can right-click on your class (in the editor or Package Explorer) and select **Refactor □Rename** to rename your class. Eclipse will make sure that all calls in your Workspace to your class or method are renamed.

The following screenshot shows how to call the Rename refactoring for a class. The cursor is positioned on the class and the context menu is activated via a right-click on the class.

The most important refactorings are listed in the following table.

Table 1. Refactoring

Refactoring	Description
Rename	Rename a variable or class
Extract Method	Creates a method based on the selected code in the editor

Extract Constant	Gives magic numbers or hard-coded strings a descriptive constant name and replaces all occurences.

Lots of refactorings are also available via the `Ctrl`+`1` shortcut (quick fix). Select a certain part of your code and press `Ctrl`+`1` to see possible refactorings for the selected position.

Eclipse has many more refactorings. The available options depend on the selection in the Java editor. In most cases you should get an idea of the performed action by the naming of the refactoring operation.

14. Exercise: Refactoring

14.1. Preparation

For the next examples change the `MyFirstClass` class to the following code.

```
package com.vogella.eclipse.ide.first;

public class MyFirstClass {

  public static void main(String[] args) {
    System.out.println("Hello Eclipse!");
    int sum = 0;
    for (int i = 1; i <= 100; i++) {
      sum += i;
    }
    System.out.println(sum);
  }
}
```

14.2. Extract method

A useful refactoring is to mark code and create a method from the selected code. Mark now the coding of the "for" loop, right click on the selection and select **Refactoring □ Extract Method**. Use calculateSum as the name of the new method.

After this refactoring the class should look like the following code.

```
package com.vogella.eclipse.ide.first;

public class MyFirstClass {

  public static void main(String[] args) {
    System.out.println("Hello Eclipse!");
    int sum = 0;
    sum = calculateSum(sum);
    System.out.println(sum);
  }

  private static int calculateSum(int sum) {
    for (int i = 1; i <= 100; i++) {
      sum += i;
    }
    return sum;
  }
}
```

14.3. Extract Constant

You can also extract strings and create constants based on the strings. Mark for this example the `"Hello Eclipse!"` string in your source code, right-click on it and select **Refactor □ Extract Constant**. Name your new constant `HELLO`.

The string is now defined as a constant.

```
package com.vogella.eclipse.ide.first;

public class MyFirstClass {
  private static final String HELLO = "Hello Eclipse!";

  public static void main(String[] args) {
    System.out.println(HELLO);
    int sum = 0;
    sum = calculateSum(sum);
    System.out.println(sum);
  }

  private static int calculateSum(int sum) {
    for (int i = 1; i <= 100; i++) {
      sum += i;
    }
    return sum;
  }
}
```

15. Eclipse Shortcuts

Eclipse provides a lot of shortcuts to work efficiently with the IDE.For a list of the most important Eclipse shortcuts please see Eclipse Shortcuts

16. Using JARs (libraries) in Eclipse

16.1. Adding a Java library to the project classpath

You can store JAR files directly in your project, and add them to the classpath which the Java compiler of Eclipse is using. To manage the classpath for your Eclipse, right-click on your project and select Properties.Under **Java Build Path** □ **Libraries** you can review and change your current classpath as depicted in the following screenshot.

JAR files can be stored outside your project or inside.To import a JAR into an existing folder, select **File** □**Import** □ **General** □**File System**.Select the Java library you want to import and select the folder, e.g.,lib, as target. To add JAR file located in the project to its classpath, right-click on the JAR file and select **Build Path □Add to Build Path**.

Alternatively, to the import approach via the menu, you can copy and paste the `jar` file into a folder.

Outside Eclipse you still need to configure your classpath for your project.

16.2. Using project dependencies

You can define in Eclipse that a project is dependent on another project. If you do this, you can use its classes in the project defining the dependency. To do this select your project, right-click on it and select Properties. Select Java Build Path and the Projects tab.

This only works within Eclipse, it allows you to develop several projects which will later be exported as JAR files together. Outside of Eclipse you need to create Java libraries for the projects and add them to the classpath.

16.3. Attach source code to a Java library

You can open any class by positioning the cursor on the class in an editor and pressing `F3`. Alternatively, you can press `Ctrl`+`Shift`+`T`. This shows a dialog in which you can enter the class name to open it.

If the source code is not available, the editor shows the bytecode of that class.

This happens, for example, if you open a class from a the standard Java library without attaching the source code to it.

To see the source code of such a class, you can attach a source archive or source folder to a Java library. Afterwards, the editor shows the source instead of the bytecode.

Attaching the source code to a library also allows you to debug this source code.

The Source Attachment dialog can be reached in the Java Build Path page of a project. To open this page, right-click on a project and select **Properties ▸ Java Build Path**. On the Libraries tab, expand the library's node, select the Source Attachment attribute and click the `Edit` button.

In the Location path field, enter the path of an archive or a folder containing the source.

The following screenshot shows this setting for the standard Java library. If you have the Java Development Kit (JDK) installed, you should find the source in the JDK installation folder. The file is typically called src.zip.

16.4. Add Javadoc to a Java library

It is also possible to add Javadoc to a library which you use. For this you need to have the Javadoc somewhere in your filesystem. To configure the location of the Javadoc for the JAR File, open the Java Build Path via a right-click on a project. Select **Properties ▸ Java Build Path**. On the Libraries tab expand the library's node, select the Javadoc location attribute and press the `Edit` button.

Enter the location to the file which contains the Javadoc.

17. Updates and installation of plug-ins

17.1. Eclipse update manager

The Eclipse IDE allows you to install and update software components.The Eclipse update functionality only downloads new or updated components.

Installable software components are located in update sites. Update sites can be located on a web server or on the file system.

17.2. Performing an update

If you are behind a network proxy, you have to configure your proxy via the Window □ Preferences □General □Network Connection preference setting.Otherwise, Eclipse may not able to reach the update sites.

To update your Eclipse installation, select **Help □Check for Updates**.The system searches for updates in the configured update sites for the installed software components.If it finds updated components, it will ask you to approve the update.

17.3. Install new functionality

To install a new functionality, select **Help** □Install New Software....

From the Work with list, select or enter a URL for the update site you want to use.Entering a new URL adds this URL automatically to the list of available update sites.

To explicitly add a new update site, press the Add... button and enter the new URL as well as a name for the new update site.

The following update sites contain the official Eclipse components.

```
# Release specific update site, e.g. for the 2020-09
release
http://download.eclipse.org/releases/2020-09

# Update site for the latest release
http://download.eclipse.org/releases/latest
```

If you select a valid update site, Eclipse allows you to select components and install them.

If you can’t find a certain component, uncheck the _Group items by category _ checkbox because not all available plug-ins are categorized. And not categorized items are only displayed if the grouping is disabled.

17.4. See the installed components

To see which components are installed, use **Help □About Eclipse SDK □Installation Details**.

17.5. Uninstalling components

If you select **Help □About Eclipse SDK** and then the `Installation Details` button, you can uninstall components from your Eclipse IDE.

17.6. Restarting Eclipse

After an update or an installation of a new software component, you should restart Eclipse to make sure that the changes are applied.

18. Eclipse Marketplace

18.1. Using the Marketplace client

Eclipse contains a client which allows installing software components from the Eclipse marketplace. The advantage of this client is that you can search for components, discover popular extensions and see descriptions and ratings.

Compared to the update manager, you do not have to know the URL for the software site which contains the installable software components.

Most Eclipse distributions contain the Marketplace client by default. You may need to install the Marketplace client software component into Eclipse before you can use it. The following screenshot shows how to install it from one of the official Eclipse update sites, via the **Help** □Install New Software... dialog.

To open the Eclipse Marketplace, select **Help** □Eclipse Marketplace....

You can use the Find box to search for components. Pressing the `Install` button starts the installation process.

18.2. Maintaining your Favorites

The marketplace client allows to install your favorite plug-ins directy. For this, go to the Eclipse Marketplace website and login with your Eclipse.org account.

Now search on the website for your favorite and press the * sign.

Afterwards, select the Favorites tab in the Eclipse Marketplace client and login to be able to install your favorites.

19. Advanced Eclipse Update manager options

19.1. Manual installation of plug-ins (dropins folder)

Eclipse plug-ins are distributed as jar files. If you want to use an Eclipse plug-in directly or do not know the update site for it, you can place it in the dropins folder of your Eclipse installation directory. Eclipse monitors this directory and during a (re-)start of your IDE, the Eclipse update manager installs and removes plug-in based on the files contained in this directory.

You should not modify the content of the Eclipse plugins directory directly. If you want to install plug-ins, put them into the dropins folder. If you want to remove it, delete the JAR from this folder.

Plug-ins are typically distributed as jar files. To add a plug-in to your Eclipse installation, put the plug-in jar file into the Eclipse dropins folder and restart Eclipse. Eclipse should detect the new plug-in and install it for you.

If you remove plug-ins from the dropins folder and restart Eclipse, these plug-ins are automatically removed from your Eclipse installation.

19.2. Exporting and importing the installed components

Eclipse allows you to export a file which describes the installed Eclipse components. During the export the user can select which components should be included into this description file.

Other users can import this description file into their Eclipse installation and install the components based on this file.

This way, Eclipse installation can be kept in sync with each other.

To export a description file, select **File □Export □Install □ Installed Software Items to File**.□Select the components which should be included in your description file.

To install the described components in another Eclipse installation, open the exported file with **File □Import □ Install □Install Software Items from File** and follow the wizard.□The wizard allows you to specify the components which should be installed.

19.3. Installing features via the command line

The Eclipse update manager has a component called director which allows you to install new features via the command line.

For example, the following command will install the components EGit, Mylyn and EMF into an Eclipse instance. You need to start this command in the command line and it assumes that you are in a directory which contains your Eclipse installation in a folder called eclipse.

```
eclipse/eclipse \
-application org.eclipse.equinox.p2.director \
-noSplash \
-repository \
http://download.eclipse.org/releases/luna \
-installIUs \
org.eclipse.egit.feature.group,\
org.eclipse.jgit.feature.group,\
org.eclipse.emf.sdk.feature.group,\
org.eclipse.mylyn_feature.feature.group,\
org.eclipse.wst.xml_ui.feature.feature.group,\
org.eclipse.mylyn.java_feature.feature.group,\
org.eclipse.mylyn.pde_feature.feature.group
```

The feature names which you need for this operation can be seen on the second page of the standard installation dialog of the Eclipse update manager.

20. Eclipse preference settings

20.1. What are preferences?

The behavior of the Eclipse IDE can be controlled via key value pairs stores as preferences settings. Each Eclipse software component can define such perferences and use the values to configure itself. This allows you for example to configure how long the Eclipse waits before the code completion or if the import statements in your source code should be automatically adjusted if you save your source code.

Which preferences are key values stored on the file system, the Eclipse IDE allows the user to configure most of these values via the preference dialog.

20.2. Opening the preference dialog

Select **Window** ▸ **Preferences** to open the preference dialog. You can use the filter box to search for specific settings.

Correctly configuring Eclipse to your needs can largely improve your productivity. Most of these preference settings are specific to your workspace but some are also valid for all workspaces.

20.3. Configuring the preference values via the plugin_customization.ini file

You can specify default values for preferences via a file which is typically called plugin_customization.ini.

In this file you enter default values for preference settings. For example, the following will setup a default type filter for the `java.awt` and `javax.swing` package.

```
org.eclipse.jdt.ui/org.eclipse.jdt.ui.typefilter.enabled=ja
va.awt.*;javax.swing.*;
```

You link to this file via your eclipse.ini file in your Eclipse installation directory.

The following example eclipse.ini links to the file and it assumes that you created the plugin_customization.ini file in the Eclipse installation directory.

```
-pluginCustomization
plugin_customization.ini
-startup
plugins/org.eclipse.equinox.launcher_1.3.200.v20160318-
1642.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.gtk.linux.x86_64_1.1.4
00.v20160504-1419
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
--launcher.appendVmargs
-vmargs
-Dosgi.requiredJavaVersion=1.8
-Xms256m
-Xmx1024m
-XX:+UseG1GC
-XX:+UseStringDeduplication
```

20.4. Identifying preference setting values

To identify a key for a certain preference setting you can export existing preference settings via the following approach.

- start a new workspace
- change the preference
- export all preferences
- search the key in the exported file

You need to remove the scope (e.g., /instance/) before copying it into the plugin_customization.ini file.

21. Exercise: Optimizing the Eclipse IDE settings

The Eclipse IDE is relatively conservative configured to avoid surprises during development. Certain settings in the Eclipse IDE allow you to use it more efficiently. In this exercise you make changes to the default Eclipse IDE settings to your personal usability with the Eclipse IDE.

If you find a setting in this exercise not working for you, you can always skip that setting. There is not a single correct setting for everyone in the world.

21.1. Link Java editor with the Project Explorer or Package Explorer view

You can synchronize the currently selected Java editor with the selection in the Project Explorer or the Package Explorer view. This gives you a clearer visibility which object you are currently editing.

Enable this by selecting the corresponding button in the view.

21.2. Automatic placement of semicolon

Eclipse can make typing more efficient by placing semicolons at the correct position in your source code.

In the Preference setting select **Java □Editor □Typing.** In the Automatically insert at correct position selection enable the Semicolons checkbox.

Afterwards, you can type a semicolon in the middle of your code and Eclipse positions it at the end of the current statement.

21.3. Auto-escape text pasted into Strings

Eclipse allows you to escape text automatically if it is pasted into a String literal. For example, you can copy HTML code and paste it into a String in your Java source. Eclipse would escape the text automatically for you.

Activate this setting via **Window □Preferences □Java □Editor □Typing □In string literals □Escape text when pasting into string literal**

Now you can paste text that should be escaped. The following code snippet shows an example for the resulting code if you paste HTML code containing a link into a string literal.

```
# paste <a href="tutorials/index.html">Tutorials</a>
# between "" of String s = ""

# results in:
String s = "<a
href=\"tutorials/index.html\">Tutorials</a>";
```

21.4. Bracket highlighting

You can configure Eclipse to highlight the matching brackets of a code block in the source code editor.

Before the change you would not see the enclosing brackets. Afterwards, they will be slightly highlighted. This helps to see in which block you are.

21.5. Always start previous launched application

Eclipse allows you to start an application via the `Run` button in the menu or via the `Ctrl`+`F11` shortcut. By default, Eclipse determines if the currently selected file is executable and try to start that. This is sometimes confusing. You can configure the Eclipse IDE to always start the last started program.

To configure this, select **Window □Preferences □Run/Debug □Launching**. In the Eclipse preferences dialog select the Launch the previously launched application option in the **Run/Debug □Launching** setting.

21.6. Filtering out certain Java packages via Type filters

To add import statements to your code, you can use the Organize Imports action (shortcut: `Ctrl`+`Shift`+`O`). If there are several alternatives, Eclipse suggests all available packages and the user has to select the right one.

The Save Actions setting can automatically organize import statements. It adds import statements automatically if there is only one possible import and removes unused ones.

The following shows the available packages for the `List` class in the Organize Imports dialog.

If you never use certain packages, for example AWT or Swing, you can exclude these packages from Eclipse via the **Window □Preferences □Java □Appearance □Type Filters** setting.

Press the `Add packages` button to add a specific package or the `Add...` button to use wildcards.The setting in the following screenshot excludes the `java.awt` and `java.swing` packages from the possible imports and other Java search functionality.

Please note that Eclipse shows (in its default configuration) only the packages that are used in the current workspace.If you want to exclude standard Java packages, you have to create at least one Java project.

21.7. Format source code, organize imports and code cleanup on save

Eclipse can perform actions during the save operation (shortcut: `Ctrl`+`S`)., e.g., format your source code, organize your imports can cleanup your code.You can find the corresponding settings under **Window □Preferences □Java □ Editor □Save Actions**.

Select that the source code should be formated and that the imports should be organized at every save action.

Import statements are only automatically created if where is one valid import. If Eclipse determines more than one valid import, it will not add import statements automatically. In this case you still need to right-click in your editor and select Source □Organize Imports (shortcut: `Shift` + `Ctrl` + `O`).

You can improve this with Filtering out certain Java packages via Type filters, as this reduces the list of possible imports.

You can also define the additional actions which are performed during save. Via the `Configure…` button you can select which one you want to activate.

21.8. Completion overwrites and insert guessed method arguments

Eclipse can override existing method calls, in case you trigger a code completion in an existing statement. Eclipse can also try to guess the correct actual parameters for a method call.

With the first setting you can override methods in the middle of a statement via the `Ctrl` + `Space` code assists shortcut.

Without this setting you would get the following result, which results in a syntax error.

With this setting you get the following result.

21.9. Auto activation key for code completion

Due to Bug 348857 this setting is not usable at the moment.

The Eclipse IDE is configured to give you automatic code completion suggestion only after the . sign. YOu can configure Eclipse to get code completion on every character. Open again the **Window □Preferences □Java □Editor □ Content Assists** preference setting and enter .abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVXYZ in the Auto activation trigger for Java.

22. Eclipse code checks and cleanup

22.1. Java Development Toolkit code checks

You can define how the Java compiler should react to certain common programming problems, e.g., you can define that an assignment of a variable which has no effect, e.g., `x=x`, causes an error in Eclipse.

22.2. Configuring the code settings

You can configure these checks in the Eclipse preferences settings via the **Java □Compiler □Errors/Warnings** entry.

22.3. Annotation-based Null analysis

You can enable annotation-based null checks in Eclipse via the setting highlighted in the following screenshot.

After enabling this setting, you can use the `@NonNull` annotation on method parameters or variable definitions to indicate that these are not allowed to be NULL. You can also use the `@Nullable` annotation to define that a variable can be NULL.

22.4. Running a code cleanup and removal of trailing whitespace

Eclipse has the option to perform cleanup actions on existing code. This includes the removal of trailing whitespace, the additional of missing annotations but also advanced cleanups like the conversion of code to Java 8 lambda expressions.

To trigger this cleanup, select the Java source file, package or project and select **Source** □Clean-up... from the context menu.

Select `Use custom profile` and press `Configure...` to configure the actions which should be performed on your code.

Ensure to unselect any cleanup action which you do not want to perform.

After finishing the configuration, press `OK` and the `Next` button in the cleanup wizard to get a preview of the changes.

23. More on preference settings

This chapter lists other useful Eclipse settings which are not directly related to Java development. It also explains how to export and import your preference settings from one workspace to another.

23.1. Configuring the editors for a file extension

The Editors which are available to open a file can be configured via **Window □Preferences □General □Editors □ File Associations**.

The `Default` button in this preference dialog allows you to set the default editor for a certain file extension. This edit is used by default, if you open a new file with this extension.

The other configured editors can be selected if you right-click on a file and by selecting **Open With**. In the sub-menu you see the available editors. The available editors depend on your Eclipse installation.

Eclipse remembers the last editor used to open a file. It uses this editor again the next time you open the file.

23.2. Export and import preference settings

You can export your preference settings from one workspace via **File □Export □General □Preferences**.

Eclipse does allow you to export some preference settings separately, but for most of them you have to select the Export all flag.

Similarly, you can import them again into another workspace via **File □Import □General □Preferences**.

23.3. Preference settings per project

You can also configure certain preference settings on a per project basis. To do this, select your project, right-click on it and select Properties. For example, on the **Java Editor □Save**

Actions you can select the Enable project specific settings checkbox to configure the save action for the current project only.

This creates a .settings folder.You can add this folder to your version control system to ensure that every developer uses the same setting.

24. Using templates and code formatters

24.1. Templates

You can create templates for Java code snippets which you can insert via via autocomplete (`Ctrl` + `Space`) in the Java code editor.For example, assume that you are frequently creating `public void name(){}` methods.You could define a template which creates the method body for you

To create a new template, select the menu **Window □ Preferences □Java □Editor □Templates**.

Press the `New` button.Create the following template.

```
public void ${cursor} () {
}
```

`${cursor}` indicates that the cursor should be placed at this position after applying the template.

In this example the name npm is your keyword for code completion.

Now, if you type npm in the Java editor and press `Ctrl` +

`Space`, you can insert your template.

24.2. Code Formatter

Eclipse allows you also to specify the settings for formatting the source code. These rules are used by Eclipse if you automatically format your source code.

You find the settings under **Window □Preferences □Java □ Code Style □Formatter**.

Press the `New` button to create a new set of formatting rules or press the `Edit` button to adjust an existing profile.

You can set the code formatter specific for a project via Right-click on the project □ Properties. This way you can ensure that everyone is using the same formatter while working on this project.

24.3. Code Templates

Eclipse can generate source code automatically. In several cases comments are added to the source code.

Select **Window □Preferences □Java □Code Style □Code Templates** to change the code-generation templates.

In the code tree you have the templates. Select, for example, menu:Code [Method Body] and press the `Edit` button to edit this template and to remove the "todo" comment.

25. Exercise: Custom code template usage

25.1. Create template for try/catch/finally

Create a template which creates the following block.

Place the cursor after the first bracket after the `try` statement.

25.2. Use template

Test your template in the Java editor and ensure that it works as expected.

26. Eclipse command line configuration

26.1. Eclipse memory and performance settings

Your Eclipse installation contains a file called eclipse.ini which allows you to configure the memory parameters for the Java virtual machine which runs the Eclipse IDE. For example, the -Xmx parameter can be used to define how large the Java heap size can get. -Xms defines the initial heap size of the Java virtual machine.

The following listing shows an example eclipse.ini file. The parameters after -vmargs configure the Java virtual machine. On a modern machine (with at least 8 Gigabyte available memory) assigning 2024 MB or more to the Java virtual machine is a good practice to run Eclipse faster.

```
-startup
plugins/org.eclipse.equinox.launcher_1.3.0.v20120522-
```

```
1813.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.gtk.linux.x86_64_1.1.2
00.v20120913-144807
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
-vmargs
-Xms512m
-Xmx2024m
-XX:+UseParallelGC
```

You can in additional also turn of class verification in the JVM. This avoids that the JVM checks if the class data which are loaded is not corrupt or invalid. This check is only really important if byte code in manipulated and adds 10-20% additional startup time. To disable this check add the -Xverify:none option option on your JVM.

These options can also be specified per invocation of the Eclipse IDE, e.g., for desktop shortcuts. For example to start Eclipse with 2GB of memory, use the following command line: `./eclipse -vmargs -Xmx2024m`.

26.2. Eclipse startup parameters

Eclipse allows you to configure it via startup parameters. This requires that you start Eclipse from the command line or that you configure your launcher links to include these parameters.

The following table shows important parameters.

Table 2. Workspace startup parameters

Parameter	Description
-data workspace_path	Predefine the Eclipse workspace

-showLocation	Enables the display of the current workspace directory in the header of the running IDE

For example, if you want to start Eclipse under Microsoft Windows using the c:\temp directory as workspace, you can start Eclipse via the following command from the command line.

```
c:\eclipse.exe -data "c:\temp"
```

Depending on your platform, you may have to put the path name into double quotes.

You find all available runtime options in the Eclipse help if you search for the "Eclipse runtime options" term.

27. Local history for files

27.1. Local history

Eclipse keeps a local history of files which have changed. Every time an editable file is saved, the Eclipse runtime updates the local history of that file and logs the changes that have been made. This local history can then be accessed and used to revert the file changes or to compare against a previous version.

27.2. Compare files based on local history

To compare the current version of a file with a local version stored by Eclipse, right-click on the file and select **Compare With** ▸ Local History... from the context menu. Eclipse opens the History view.

```
If you double-click on an older version of the file, the
```

```
_Compare_
view
shows the differences as depicted in the following
screenshot.
```

27.3. Replace files based on local history

You can replace files based on the local history. Right-click on the file and select **Replace With** □Local history... to start this action.

28. Organizing your workspace with working sets

You will create more and more projects in your development career. Therefore, the data in your workspace grows and it is hard to find the right information. The Eclipse IDE allows you to organize your project into working sets so that you can hide certain resources.

You can use working sets to organize your displayed projects / data. To set up your working set, select the **Package Explorer** □**open the drop-down menu** □Select Working Set...

Press the `New` button on the following dialog to create a working set.

On the next dialog select Resource, press the `Next` button. Select the projects you would like to see and give it a name.

You can now filter the displayed files in the Package Explorer based on the created working set.

You can also use the working set to structure your projects in your workspace. For this, select Working Sets from the context menu of the Package Explorer view.

29. Tasks

29.1. Task management

You can place markers in the code which you can later access via the Task view.

You can use `// TODO`, `// FIXME` or `// XXX` tags in your code to add task reminders.

This indicates a task for Eclipse. You find those in the Task view of Eclipse. Via double-clicking on the task, you can navigate to the corresponding code.

You can open this view via **Window □Show View □Tasks**.

For example, add a TODO to your `MyFirstClass` class to see it in the Tasks view.

```
package com.vogella.eclipse.ide.first;

public class MyFirstClass {

  private static final String HELLO = "Hello Eclipse!";
```

```
    public static void main(String[] args) {
        // TODO Provide user interface
        System.out.println(HELLO);
        int sum = 0;
        sum = calculateSum(sum);
        System.out.println(sum);
    }

    private static int calculateSum(int sum) {
        for (int i = 0; i <= 100; i++) {
            sum += i;
        }
        return sum;
    }
}
```

Close the editor for the `MyFirstClass` class.If you now double-click on the tasks, the Java editor opens again and the TODO comment is selected.

TIP:The Task view shows only the tasks from the currently open projects.

29.2. Own tags

You can also define your own tags in the Eclipse preferences via **Window □Preferences □Java □Compiler □Task Tags**.

29.3. Mylyn

A more advanced tasks management system is available with the Mylyn plug-in.

30. Eclipse online resources

30.1. Online documentations

The Eclipse help system is available from within your Eclipse installation as well as online.

With your running Eclipse IDE you can access the online help via **Help □Help Contents.** This will start a new window which shows you the help topics for your currently installed components.

You find the online help for the current release of the Eclipse IDE under the following URL: Eclipse online help. The online help is version-dependent and contains the help for all Eclipse projects of the simultaneous release.

30.2. Web resources

The Eclipse webpage also contains a list of relevant resources about Eclipse and Eclipse programming. You find these resources under the following link: Eclipse resources and Eclipse corner wiki.

You also find lots of tutorials about the usage of the Eclipse IDE from the vogella GmbH on the following webpage: vogella Eclipse IDE tutorials.

Information about Eclipse plug-in and RCP development from the vogella GmbH can be found on the following webpage: Eclipse Plug-in and RCP tutorials.

31. Reporting Eclipse bugs and asking questions

32. Asking (and answering) questions

Due to the complexity and extensibility of Eclipse, you will need additional resources to help you solve your specific problems. Fortunately, the web contains several resources which can help you with your Eclipse problems.

Currently, the best places to find, ask and answer questions are the Eclipse forums and Stack Overflow. Try to stay polite with your postings, as the Eclipse community values polite behavior.

The Eclipse forums offer several topic-specific forums in which you can post and answer questions. To post or to answer questions in the Eclipse forums, you need a valid user account in the Eclipse bug tracker. Stack Overflow also requires a user account and its community is very active. Stack Overflow allows to tag questions with the relevant keyword, e.g., Eclipse and people search for them or subscribe to these theses.

Ensure that you search the forums and mailing lists for solutions for your problem. Somebody else might has asked the same question earlier and the answer is already available.

33. Eclipse bug reports and feature requests

33.1. Reporting bugs and feature requests

If you encounter a problem with the Eclipse IDE or think about a potential improvement for it, you should report this to the Eclipse project. The Eclipse bug and feature tracker is using the open source Bugzilla project from Mozilla. In this system, you enter Eclipse error reports. You can also request new features or improvements of existing features.

Most Eclipse projects receive lots of bug and feature requests. So if you want something fixed or enhanced you may have to provide a Gerrit review for it. If the Eclipse developer

> sees that you try to fix the problem yourself, you typically receive more support from them.

33.2. Using the Eclipse bugzilla system

This bug tracker can be found under [Eclipse Bugzilla](). Here you can search for existing bugs and review them.

To participate actively in the Eclipse bug tracker, you need to create a new account. This can be done by clicking the Create a New Account link.

Once you have a user account, you can login to the Eclipse bug tracker. This allows you to comment on existing bugs and report new ones. The user data for the all Eclipse sites are the same, i.e, the forum, marketplace, bug tracker, etc. Only for the Gerrit access, different user data is used.

As example you can report bugs for the Eclipse platform via the following link: [Bug report for the Eclipse platform]().

33.3. Eclipse bug priorities

The Eclipse Bugzilla system allows you and the Eclipse committer to enter the bug priority. But overall, it is up to each project do decide how they handle bugs so some variation from project to project will occur. The following rules can be used as guideline.

Table 3. Bug priorities

Priority	Description
blocker	The bug blocks development or testing of the build and no workaround is known.
critical	Implies "loss of data" or frequent crashes or a severe memory leak.

major	Implies a "major loss of function".
normal	This is the default value for new bug reports. Implies some loss of functionality under specific circumstances, typically the correct setting unless one of the other levels fit.
minor	Something is wrong, but doesn’t affect function significantly or other problem where easy workaround is present.
trivial	This describes a cosmetic problem like misspelled words or misaligned text, but doesn’t affect function.
enhancement	Represents a request for enhancement (also for "major" features that would be really nice to have).

34. Using Eclipse Bugzilla

In this exercise you use the Bugzilla system to review some of the Eclipse platform bugs. No action is excepted from you, but if you find an updated bug, you should update the bug report and describe that the problem is solved.

This exercise uses the Eclipse platform as example but you can use any Eclipse project of your choice.

34.1. Run Bugzilla query

Open to Eclipse Bugzilla and select the `Search` button. Select the Advanced Search tab and search for **Eclipse ▯Platform ▯ UI** for all bugs in status NEW, ASSIGNED, UNCONFIRMED and REOPENED.

In most cases Eclipse project have tons of unsolved bugs. If you are looking for existing software bugs, it is recommended to look at the latest bugs, e.g., the bugs which have been recently updated.

35. Next steps

To learn how to debug Eclipse Java programs, you can use https://www.vogella.com/tutorials/EclipseDebugging/article.html - Eclipse Debugging.

To learn Java Web development, you can use with https://www.vogella.com/tutorials/EclipseWTP/article.html - Servlet and JSP development. If you want to develop rich stand-alone Java clients, you can use ulink url="https://www.vogella.com/tutorials/RichClientPlatform/article.html - Eclipse RCP. You can also extend Eclipse with https://www.vogella.com/tutorials/EclipsePlugin/article.html - Eclipse Plug-ins.

Good luck in your journey of learning Java!

36. Eclipse Resources

36.1. Eclipse

Eclipse IDE book from Lars Vogel

Eclipse Homepage

36.2. Eclipse Bug Tracker and Eclipse forum

Eclipse Forum for asking questions and providing feedback

Eclipse Bug Tracker for reporting errors or feature requests

Last updated 13:10 02. Oct 2020
Change your Cookies Preferences