



Woodland
Academy

7.9 eclipseの共通機能

- GitHubの連携
- デバッグ機能



Shape Your Future

目次

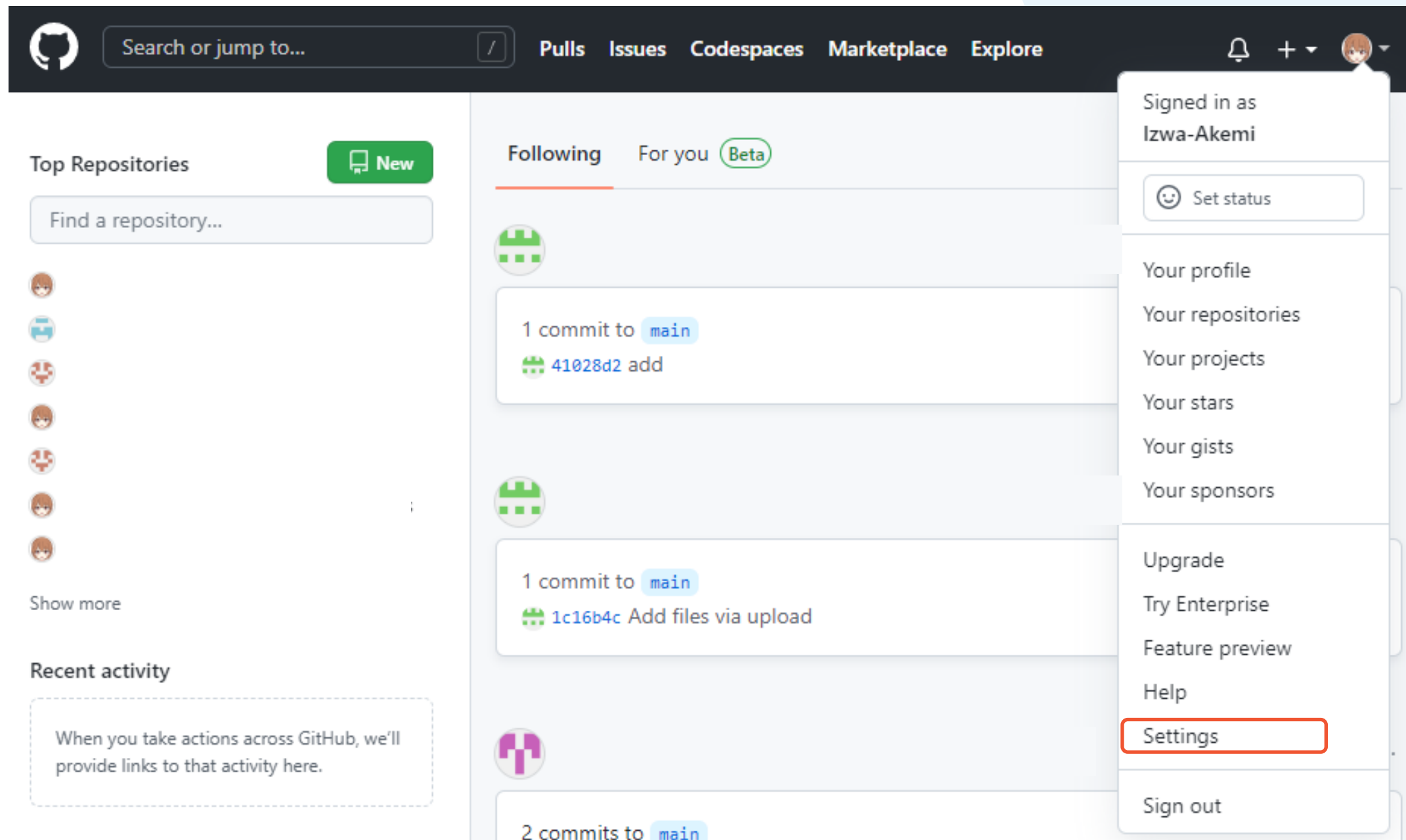
1 **GitHubの連携**

2 **デバック機能**



GitHubのアクセストークンの設定

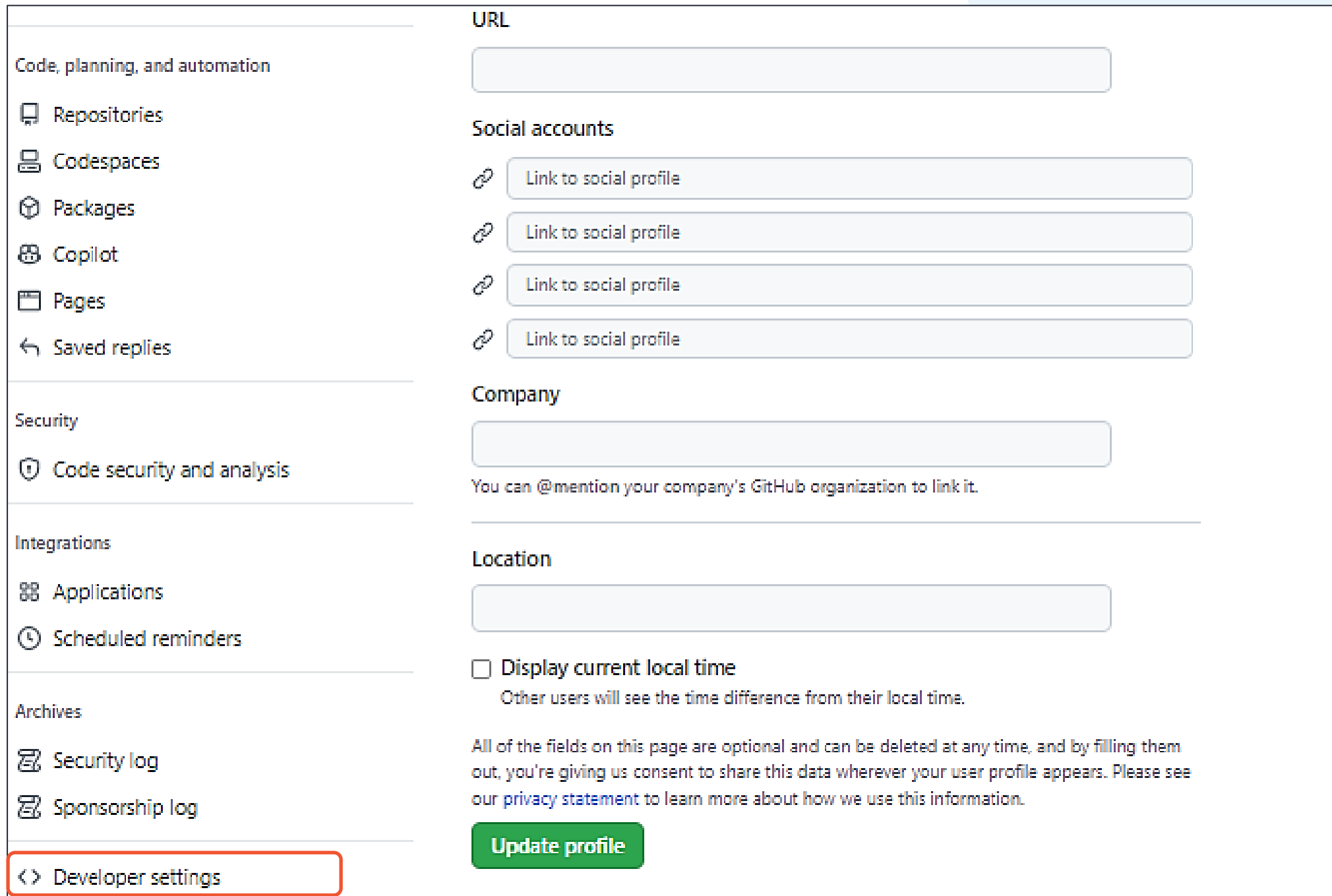
- GitHubを開いて、メニューから『**Setting**』を選択する



次へ



- 左のメニューをスクロールし、『Developer settings』を選択する



Code, planning, and automation

- Repositories
- Codespaces
- Packages
- Copilot
- Pages
- Saved replies

Security

- Code security and analysis

Integrations

- Applications
- Scheduled reminders

Archives

- Security log
- Sponsorship log

<> Developer settings

URL

Social accounts

- Link to social profile
- Link to social profile
- Link to social profile
- Link to social profile

Company

You can @mention your company's GitHub organization to link it.

Location

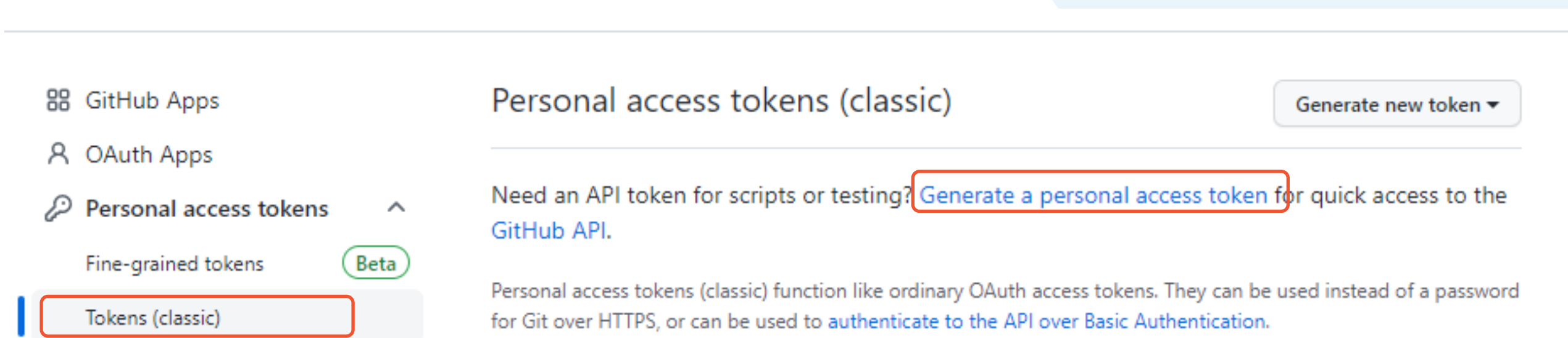
☐ Display current local time

Other users will see the time difference from their local time.

All of the fields on this page are optional and can be deleted at any time, and by filling them out, you're giving us consent to share this data wherever your user profile appears. Please see our [privacy statement](#) to learn more about how we use this information.

Update profile

- Tokens(classic)を選択し、『**Generate a personal access token**』のリンクを選択する



The screenshot shows the GitHub 'Personal access tokens (classic)' page. On the left sidebar, the navigation menu includes 'GitHub Apps', 'OAuth Apps', 'Personal access tokens' (which is expanded to show 'Fine-grained tokens' with a 'Beta' badge and 'Tokens (classic)' which is highlighted with a red box), and 'Tokens (classic)'. The main content area is titled 'Personal access tokens (classic)' and has a 'Generate new token' button. Below the title, it says 'Need an API token for scripts or testing?' followed by a blue link 'Generate a personal access token' which is highlighted with a red box. The text continues: 'for quick access to the GitHub API.' Below this, it explains: 'Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.'

GitHubのアクセストークンの設定



New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

use Eclipse

What's this token for?

Expiration *

No expiration ⚙ The token will never expire!

GitHub strongly recommends that you set an expiration date for your token to help keep your information secure. [Learn more](#)

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

必要項目を入力する

- Note: パーソナルアクセストークンの利用用途を記述
- Expiration: パーソナルアクセストークンの利用期限を設定
- Select scopes: このパーソナルアクセストークンがもつ権限をセット。Eclipseからリポジトリ操作を行いたいので repo に『✓』をいれる

GitHubのアクセストークンの設定

- 必要項目を入力及び選択後、『Generate token』をクリック

<input type="checkbox"/> admin:gpg_key	Full control of public user GPG keys
<input type="checkbox"/> write:gpg_key	Write public user GPG keys
<input type="checkbox"/> read:gpg_key	Read public user GPG keys
<input type="checkbox"/> admin:ssh_signing_key	Full control of public user SSH signing keys
<input type="checkbox"/> write:ssh_signing_key	Write public user SSH signing keys
<input type="checkbox"/> read:ssh_signing_key	Read public user SSH signing keys

Generate token Cancel

- 生成されたアクセストークンは**1度きりしか表示されない**ので、Eclipseに設定する前に**一時的にメモしておく**。

Settings / Developer settings

GitHub Apps OAuth Apps Personal access tokens ^

Fine-grained tokens (Beta) Tokens (classic)

Personal access tokens (classic) Generate new token Revoke all

Tokens you have generated that can be used to access the GitHub API.

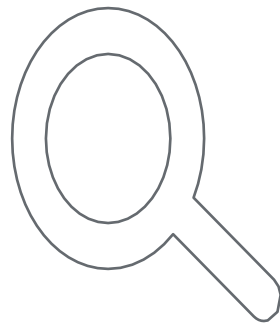
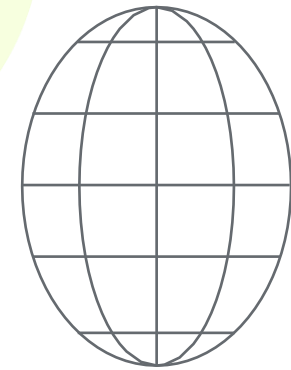
Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp_Ub9yGwVJSJq0Lv1sOp6iLg0Tqu1hba0eCMe0 Delete

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

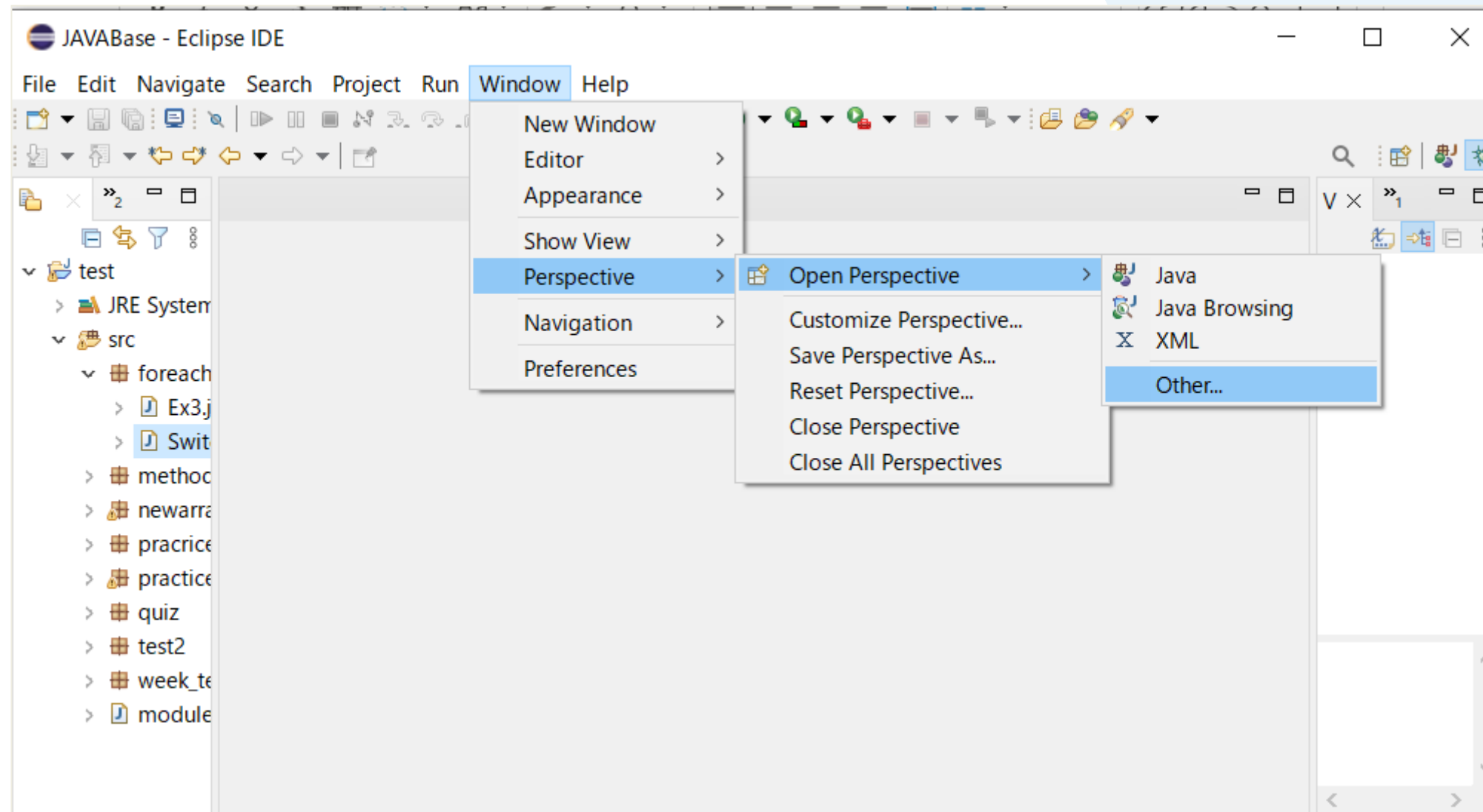


Q&A



GitHubと連携のするための準備

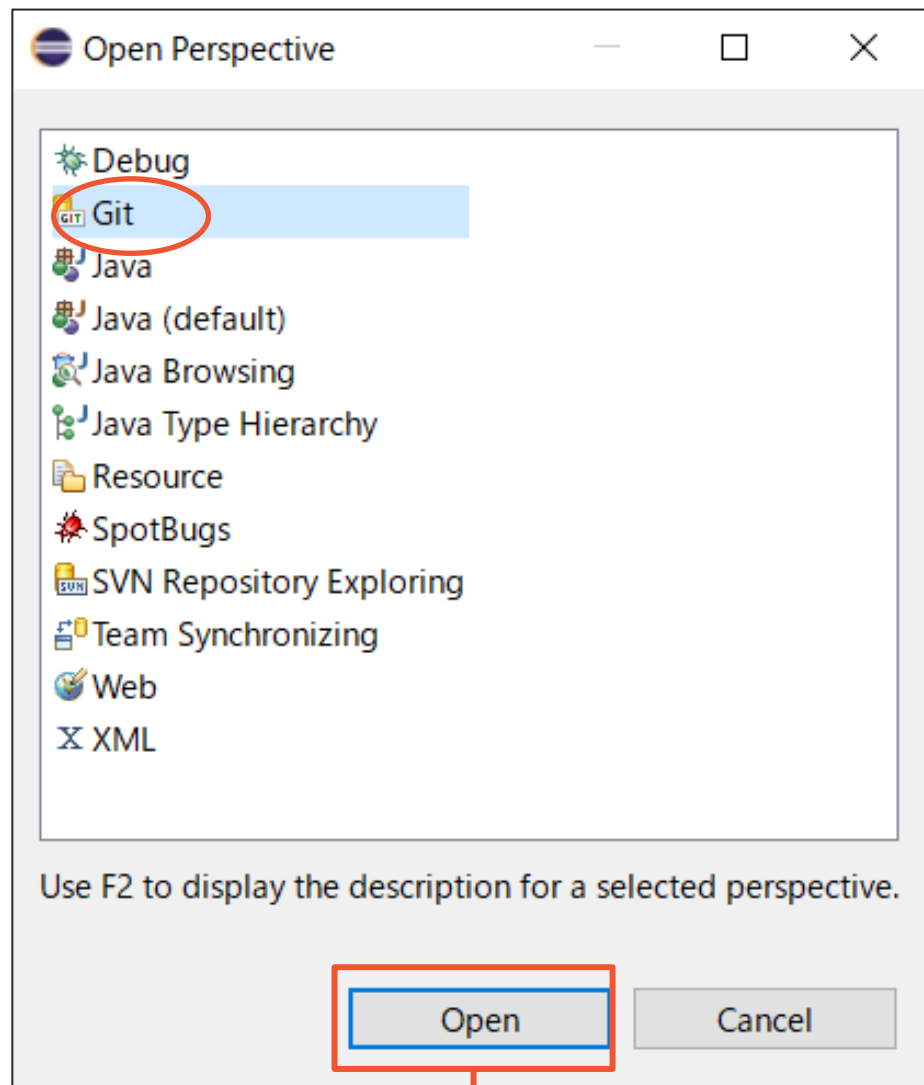
- Eclipseのメニューで『**window⇒Perspective⇒other**』を選択。



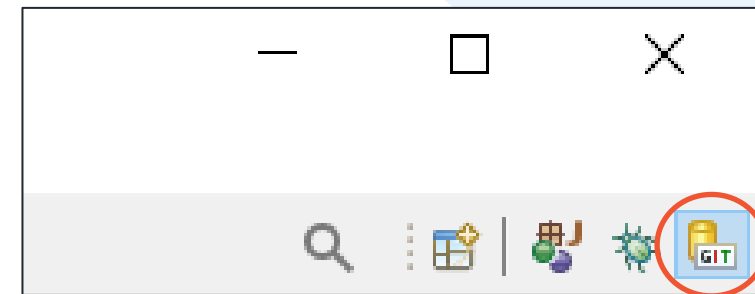
次へ

GitHubと連携のするための準備

- 開いたウィンドウで『Git』を選択して『Open』を押せば、Eclipseに『Git Perspective』を追加することができる。



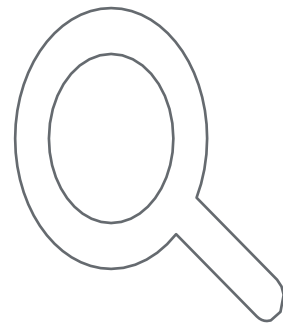
『Git』を選択して
『Open』をクリック



Eclipse右上のアイコンから、いつでもGit Perspectiveに切り替えられるようになります。

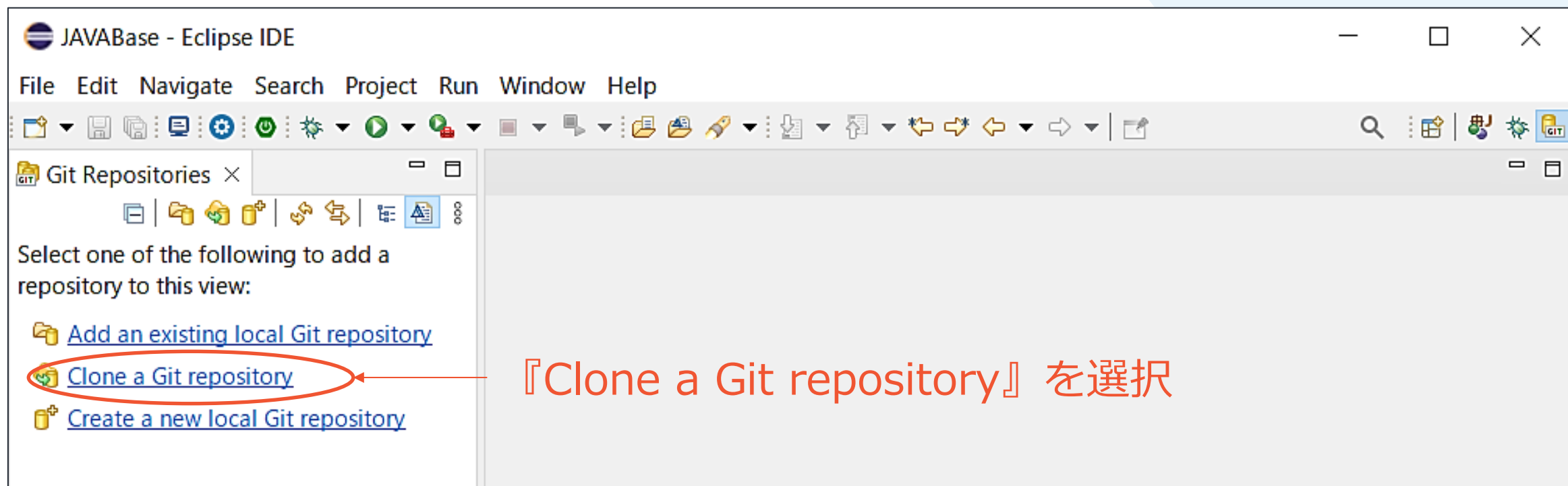


Q&A



GitHubからクローンを作成する方法

- Gitパーспекティブを開いた状態で、左側のパネルにある『Clone a Git repository』のメニューをクリックします。



GitHubからクローンを作成する方法

- 各項目に必要な内容を入力し、『Next』をクリック

Local Codespaces

Clone

HTTPS SSH GitHub CLI

`https://github.com/Izwa-Akemi/object2-3.git`

Use Git or checkout with SVN using the web URL.

Clone Git Repository

Source Git Repository

Enter the location of the source repository.

Location

URI: `https://github.com/Izwa-Akemi/object2-3.git` Local Folder... Local Bundle File...

Host: `github.com`

Repository path: `/Izwa-Akemi/object2-3.git`

Connection

Protocol: `https`

Port:

Authentication

User: `Izwa-Akemi`

Password:

☐ Store in Secure Store

< Back Next > Finish Cancel

HostとRepository pathはURIを入力すると自動的に入力される

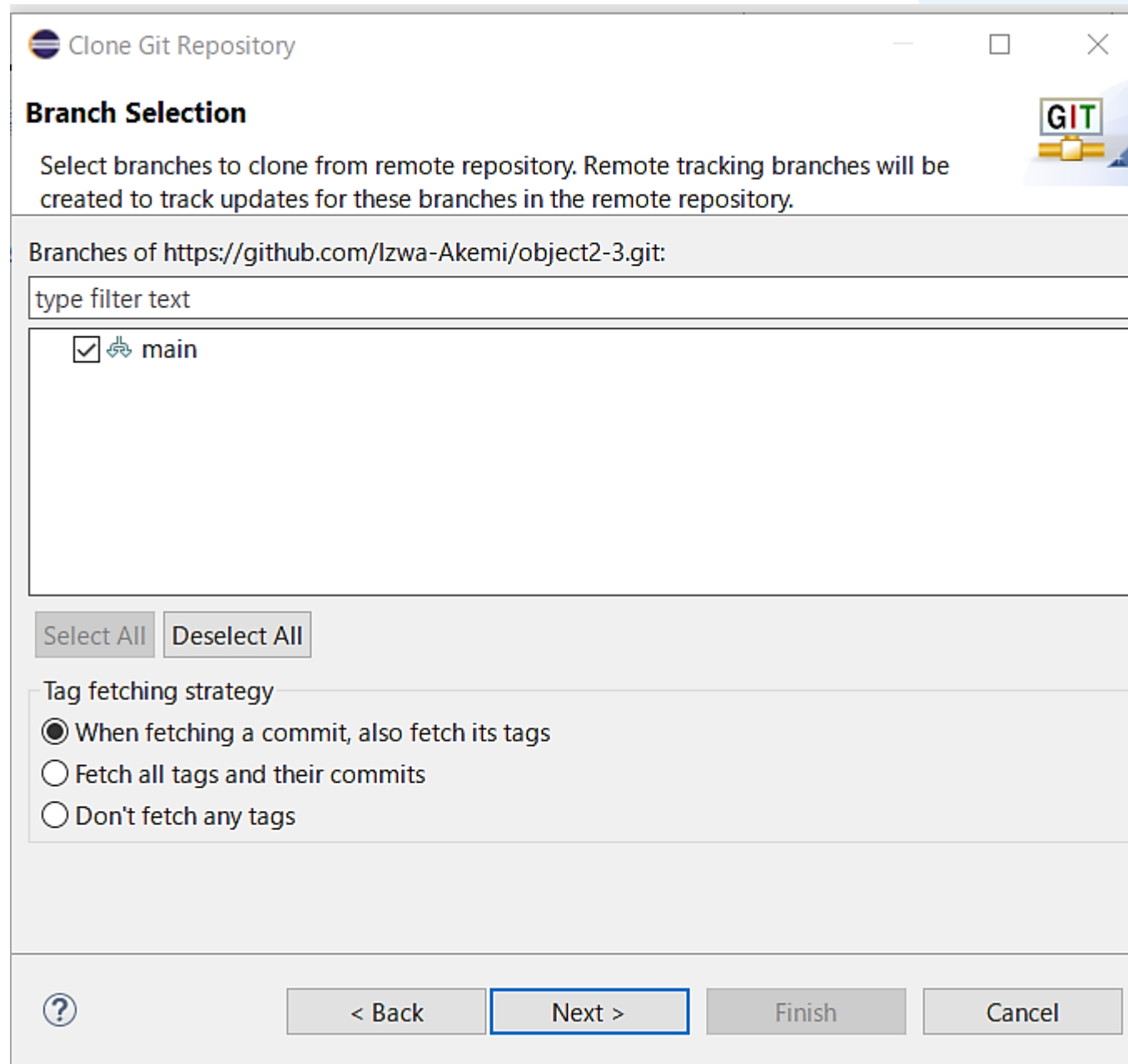
GitHubのユーザー名を入力

アクセストークンを入力

次へ

GitHubからクローンを作成する方法

- 取り込むブランチを選択し、『Next』をクリック



次へ



GitHubからクローンを作成する方法

- クローンの保管場所となるディレクトリーを指定します。ここは特に必要がなければデフォルトのままで問題ありません。最後は、『**Finish**』をクリックします。

Clone Git Repository

Local Destination
Configure the local storage location for object2-3.

Destination

Directory: Browse

Initial branch:

☐ Clone submodules

Configuration

Remote name:

Projects

☐ Import all existing Eclipse projects after clone finishes

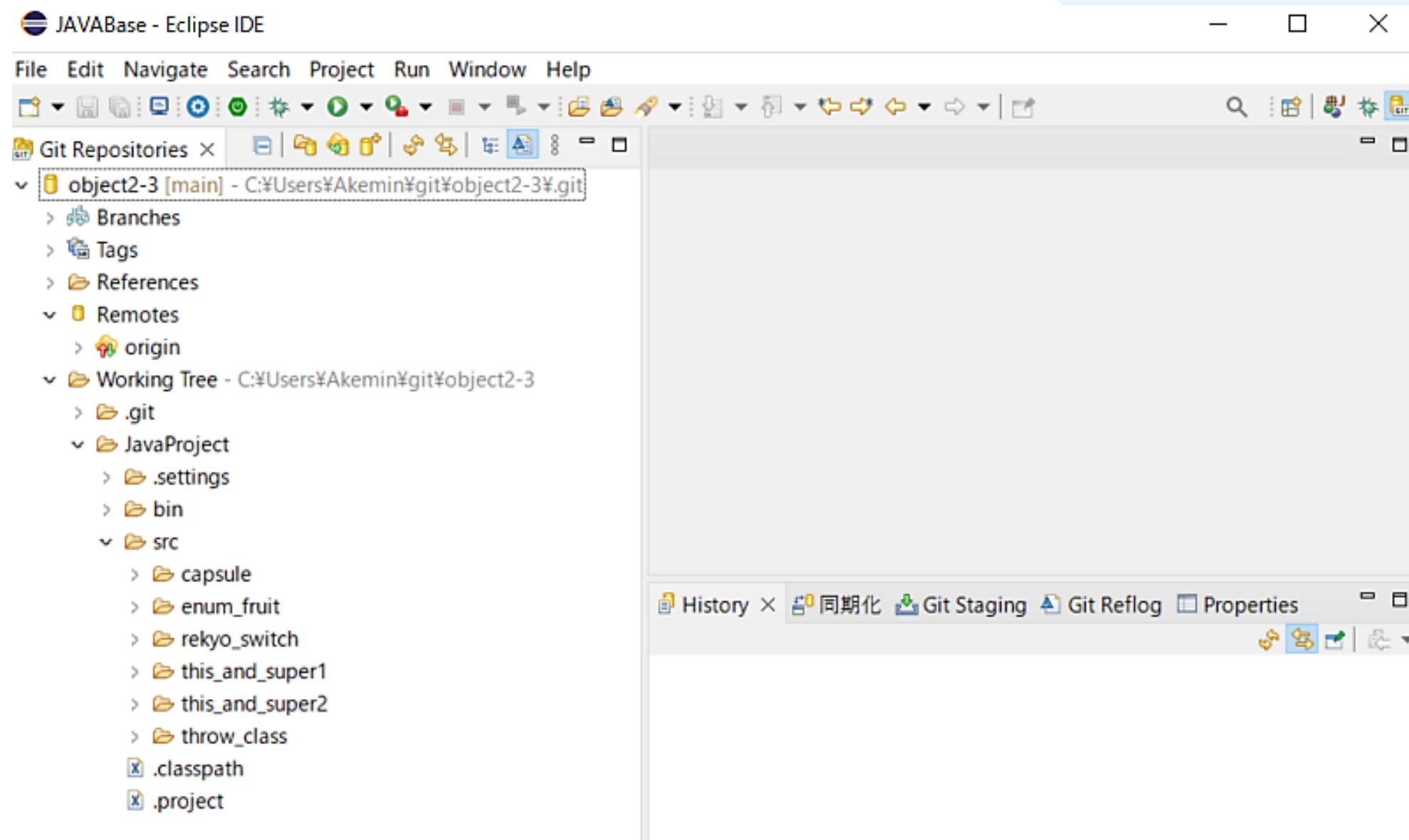
Working sets

☐ Add project to working sets New...

Working sets: Select...

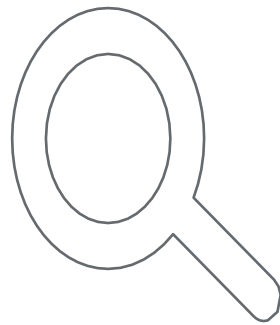
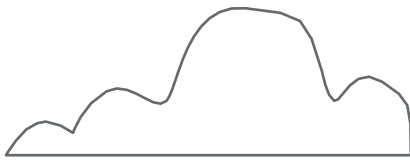
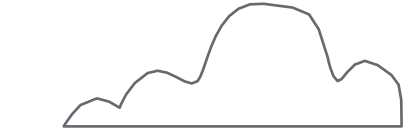
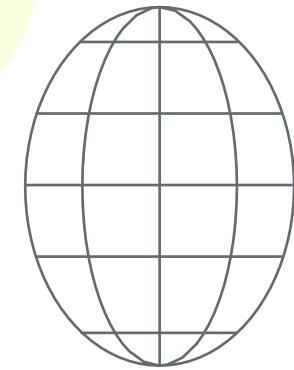
GitHubからクローンを作成完了

- クローンが完了すると、ローカルリポジトリが作成され、GitHubのリポジトリに登録されているソースコードの『クローン』が、ご自身のEclipse上で編集できるようになっています



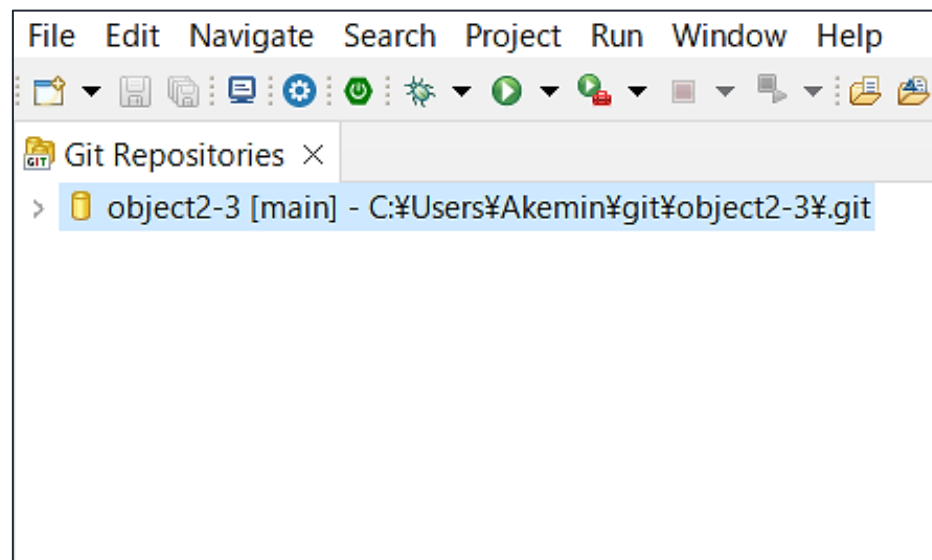


Q&A

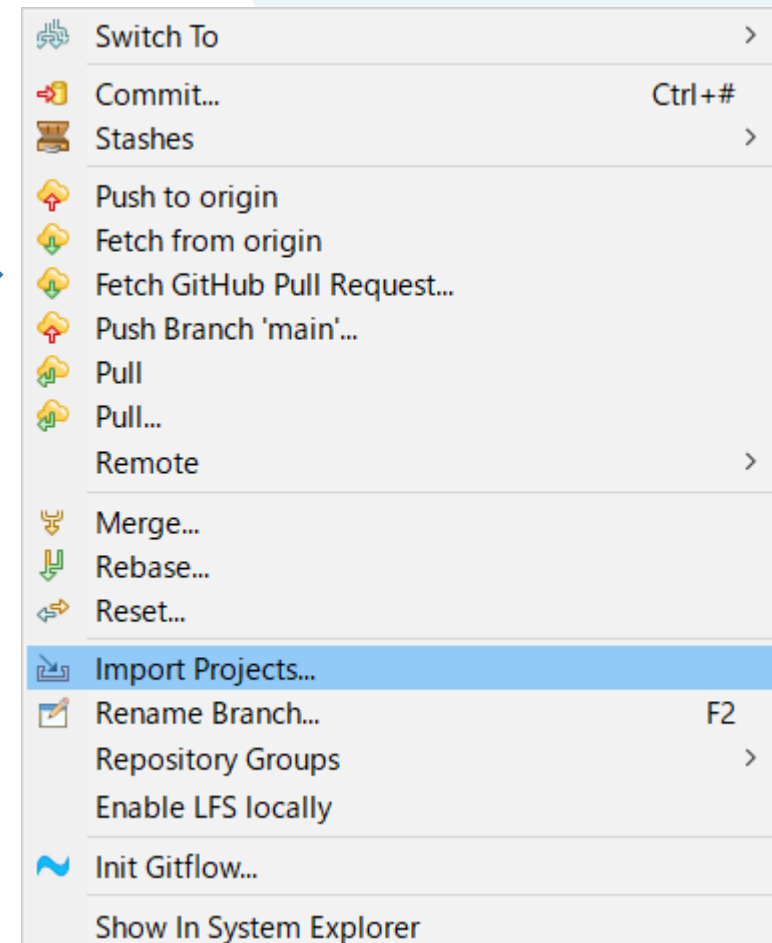


クローンからプロジェクトを作成する方法

- Gitパーспекティブを開いた状態で、インポートしたいクローンを『**右クリック⇒Import Project**』を選択します。



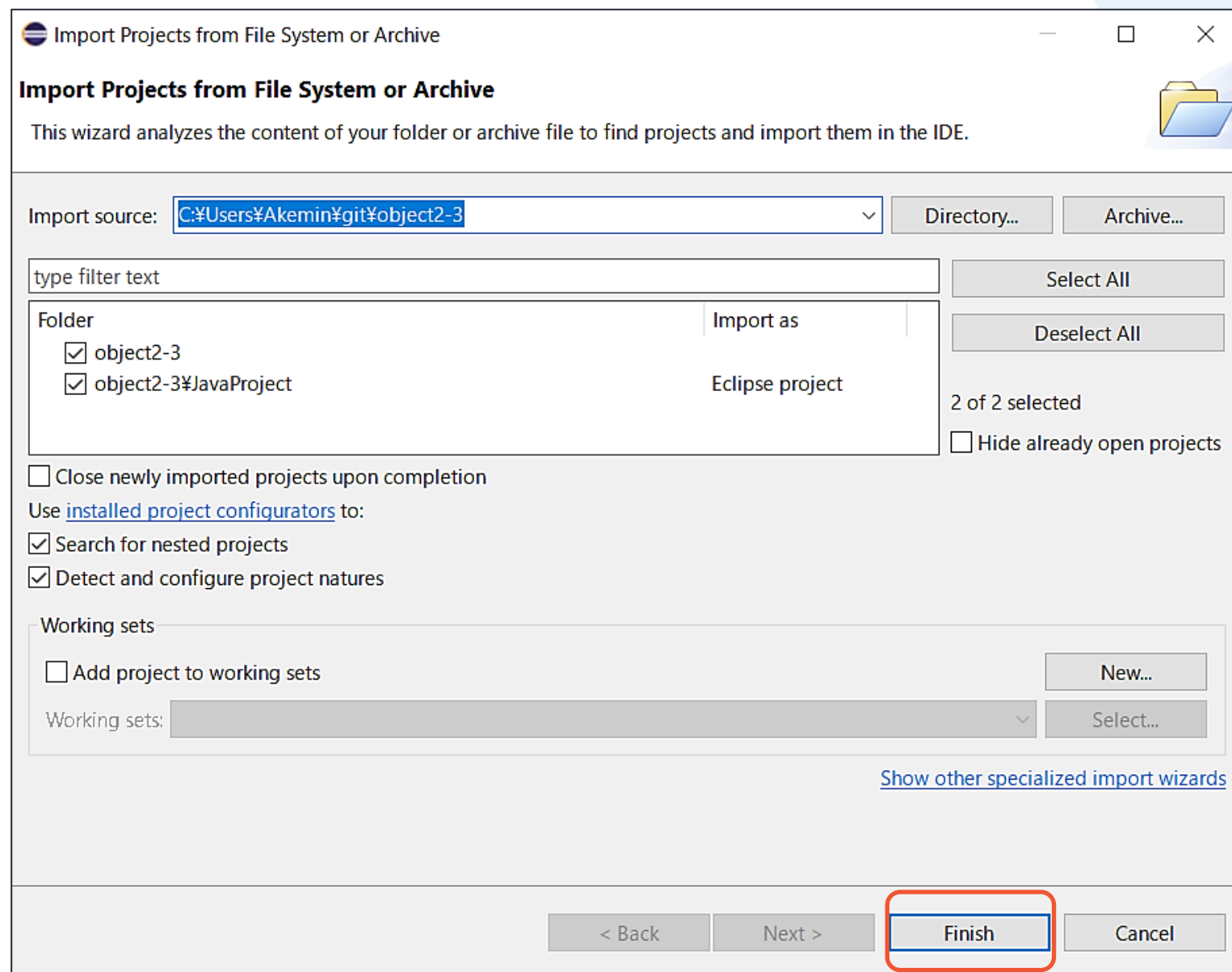
右クリック



次へ

クローンからプロジェクトを作成する方法

- 開いたウィンドウで設定を行い、インポートを完了させます。
ここでは特別な設定は必要ありませんので、そのまま『**Finish**』をクリックしましょう。

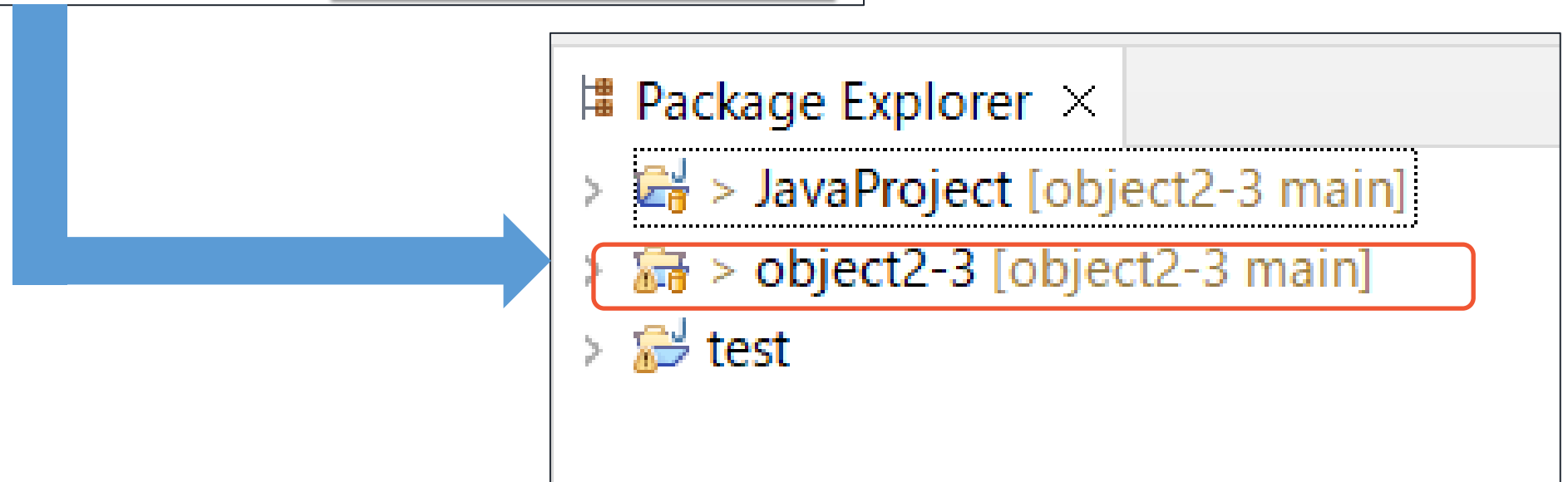
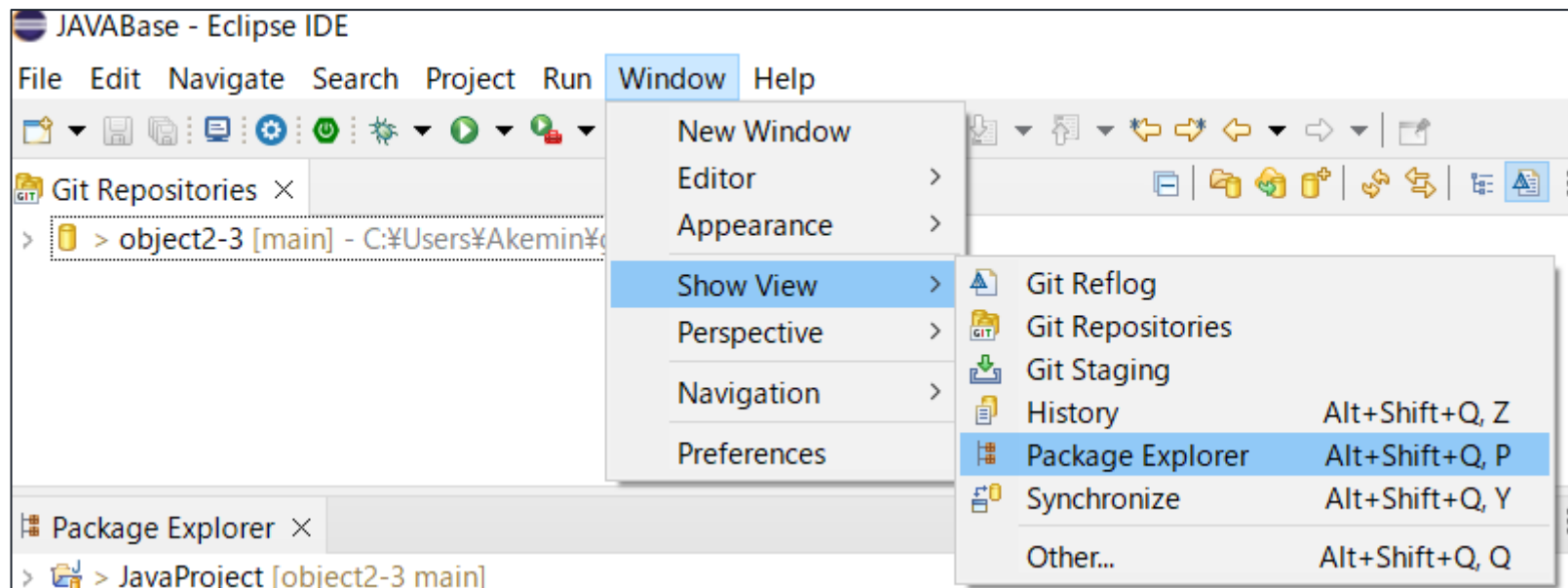


次へ



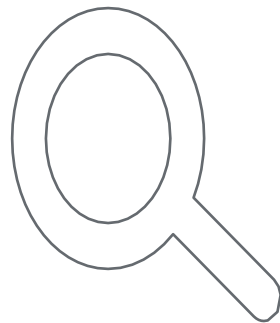
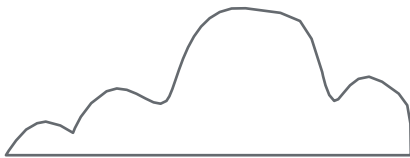
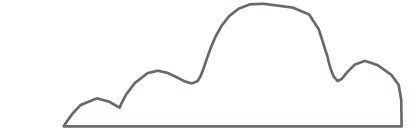
プロジェクトの確認

- メニューの『**Window**』から『**Show View**』次に『**Package Explorer**』を選択すると、プロジェクトに追加できたかを確認できる。





Q&A



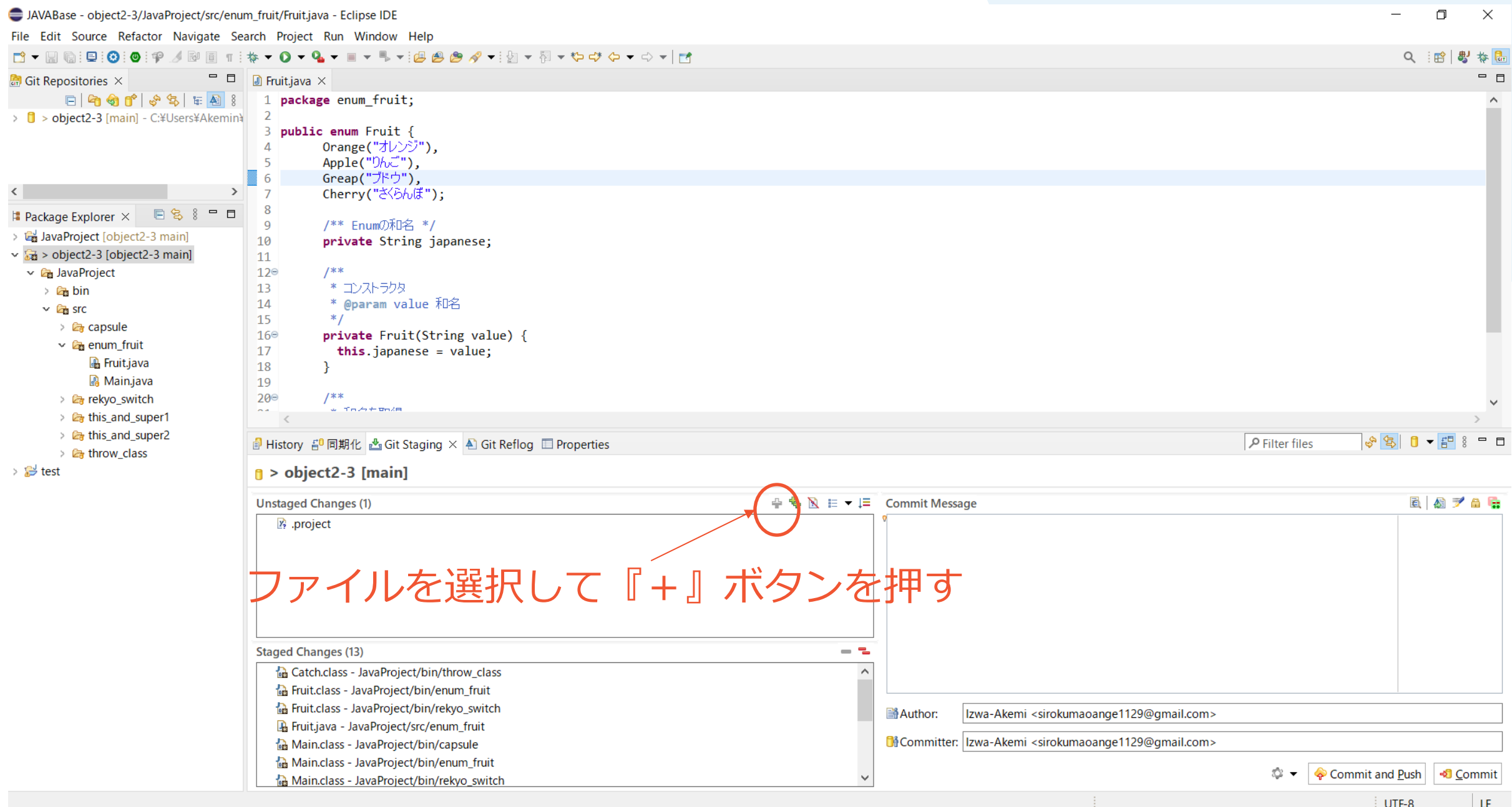
リポジトリにコミット・プッシュする方法

- まずは、ステージングするファイルを選択する
- プロジェクト名を『**右クリック⇒Team⇒Commit**』という様に選択して『**Gitステージング**』のメニューを開く。

The screenshot illustrates the workflow for committing and pulling code in an IDE. On the left, the file explorer shows a project structure with a file named `Fruit.java` selected. A red box highlights the selection, and a red arrow labeled "右クリック" (Right Click) points to the context menu. The context menu is open, showing options like "Copy", "Paste", "Delete", "Refresh", and "Team". The "Team" option is highlighted at the bottom. A second context menu is shown for the "Team" option, listing Git actions such as "Commit...", "Stashes", "Push to origin", "Fetch from origin", "Fetch GitHub Pull Request...", "Push Branch 'main'", "Pull", "Pull...", "Remote", "Switch To", "Advanced", "Synchronize Work...", "Merge Tool", "Merge...", "Rebase...", "Reset...", "Create Patch...", "Apply Patch...", "Add to Index", "Ignore", "Show in History", "Show in Repositories View", "Upgrade Projects...", "Init Gitflow...", and "Disconnect". Two callouts provide additional instructions: a red callout points to the "Pull" option with the text "ここからpullすることができます。" (You can pull from here.), and a blue callout points to the "Switch To" option with the text "ブランチを切り替えることができます。" (You can switch branches.).

リポジトリにコミット・プッシュする方法

- メニューを選択するとEclipseの下部に『Gitステージング』のパネルが表示されますので、続けてステージングするファイルを選択する。



リポジトリにコミット・プッシュする方法

- ステージングするファイルの選択、コミット・メッセージの入力が終わったら、最後にコミット方法を選択して実行します。
- 今回は、『**commit**』を選択してください。

History Synchronize Git Staging × Git Reflog Properties Filter files

> object2-3 [main]

Unstaged Changes (1)

> Fruit.java - JavaProject/src/enum_fruit

Commit Message
列挙子を追加

コミットメッセージを記載

Staged Changes (14)

- .project
- Catch.class - JavaProject/bin/throw_class
- Fruit.class - JavaProject/bin/enum_fruit
- Fruit.class - JavaProject/bin/rekyo_switch
- Fruit.java - JavaProject/src/enum_fruit
- Main.class - JavaProject/bin/capsule
- Main.class - JavaProject/bin/enum_fruit

Author: _____

Committer: _____

コミット及びプッシュ

Commit and Push

Commit

コミット

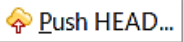

リポジトリにコミット・プッシュする方法

- 『Push Head』 ボタンをクリック
- ログインを求められた際には、GitHubのユーザー名とアクセストークンを入力
- pushしたいブランチを選択後、『push』をクリック

Commit message

Author:

Committer:



 Push HEAD...  Commit

Push Branch main

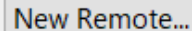
Push to branch in remote

Select a remote and the name the branch should have in the remote.

Source:

 main  52ff93b 列挙子を追加

Destination:

Remote: 


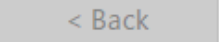
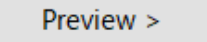
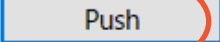
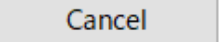
Branch:

☒ Configure upstream for push and pull

When pulling:

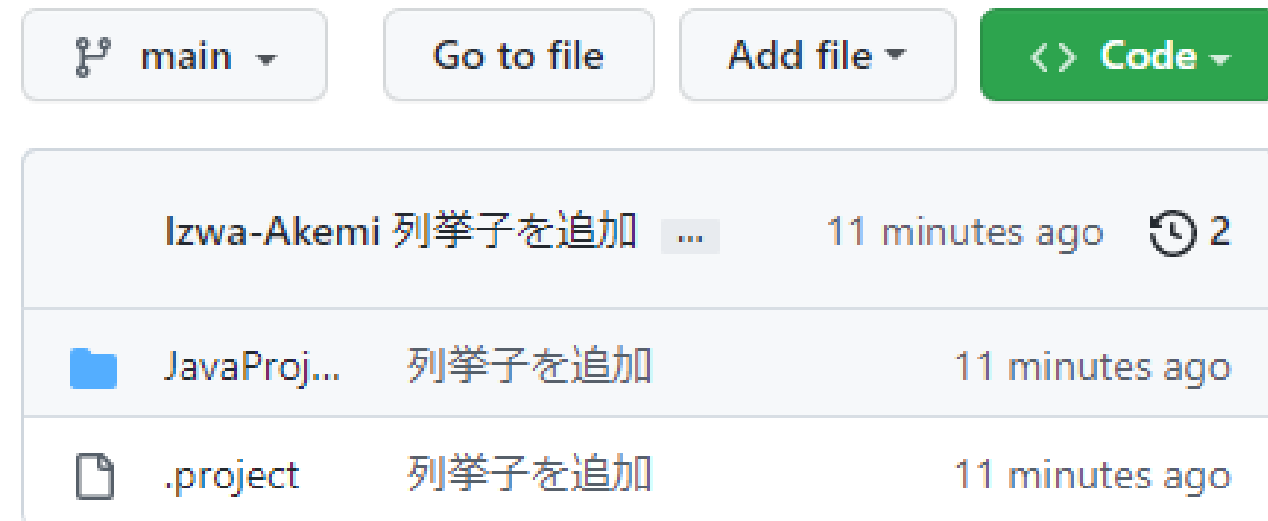
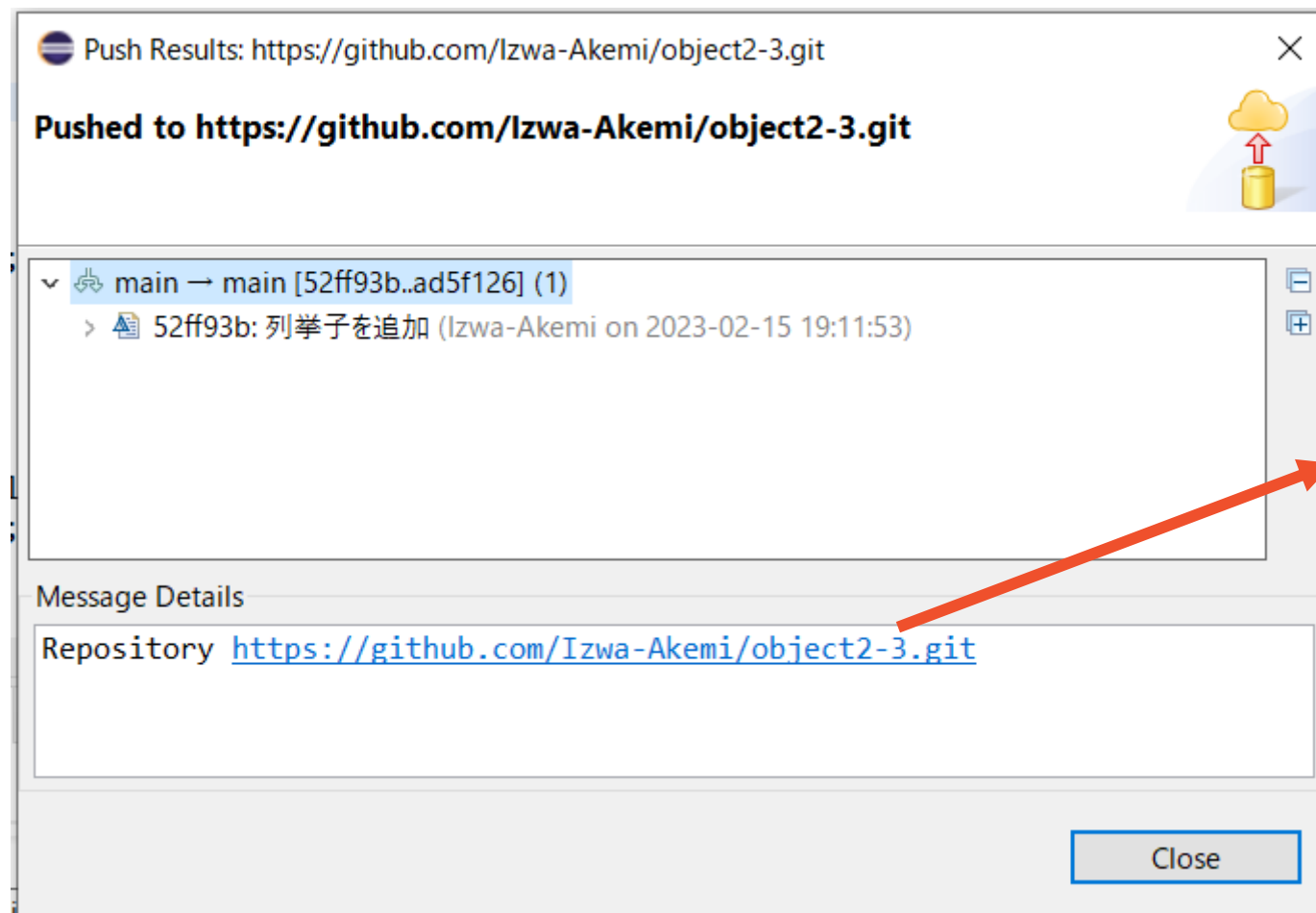
☐ Force overwrite branch in remote if it exists and has diverged

Show [advanced push](#) dialog

リポジトリにコミット・プッシュする方法

- Pushを行うと結果が出てくるため、その画面でブランチとコミットメッセージを確認することができる。
- リンクを押下して、GitHubに反映されているかを確認する。



目次

1 GitHubの連携

2 デバック機能

デバック機能

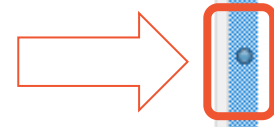
- デバック機能を紹介する前に、以下のソースをプロジェクトに追加してください。

```
1 public class DebugPractice {  
2  
3     public static void main(String[] args) {  
4  
5         String s1 = getBanana();  
6         System.out.println(s1 + "が食べたい");  
7  
8         String s2 = getStrawberry();  
9         System.out.println(s2 + "が食べたい");  
10    }  
11  
12    public static String getBanana() {  
13        return "バナナ";  
14    }  
15  
16  
17    public static String getStrawberry() {  
18        return "イチゴ";  
19    }  
20 }
```

ブレークポイントを張る

- デバッグを実行する前に、処理を止めたい行の左側をダブルクリックする。

デバッグをしたい行の
左側をダブルクリック



青丸→「ブレークポイント」

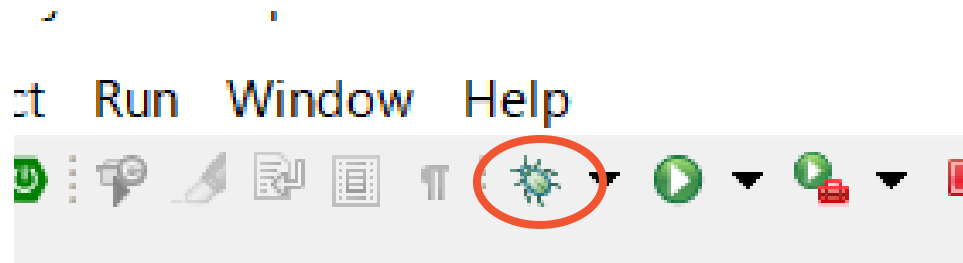
```

1 package sample;
2
3 public class DebugPractice {
4
5     public static void main(String[] args) {
6
7         String s1 = getBanana();
8         System.out.println(s1 + "が食べたい");
9
10        String s2 = getStrawberry();
11        System.out.println(s2 + "が食べたい");
12    }
13
14    public static String getBanana() {
15        return "バナナ";
16    }
17
18
19    public static String getStrawberry() {
20        return "イチゴ";
21    }
22 }

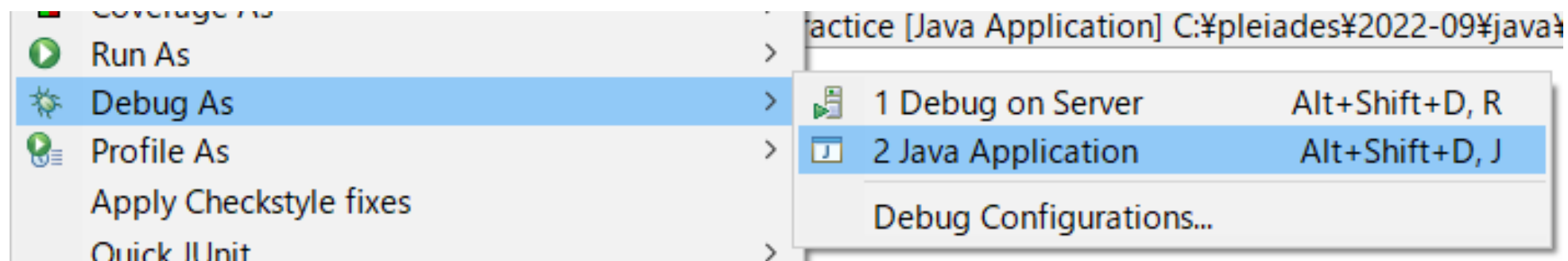
```

デバッグ実行開始

- デバッグ実行するためには、虫アイコンをクリックします。

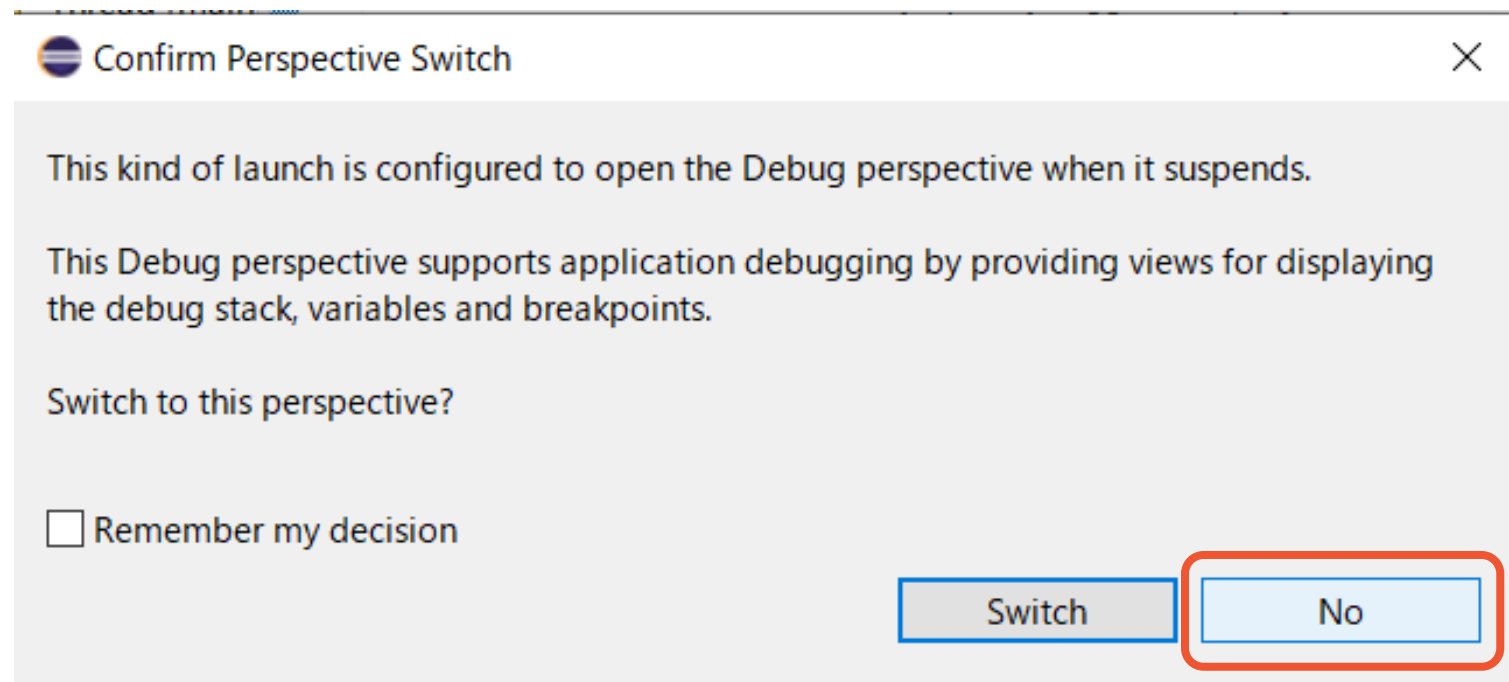


- ファイルを右クリックし、『Debug As』⇒『Java Application』からでも実行できます。



パースペクティブ切り替えの確認

- 最初に「パースペクティブ切り替えの確認」ダイアログが出ますので、「No」ボタンをクリックします。
- これで、デバッグ実行の時にウィンドウ配置が変わらなくなります。
- （デバッグ実行時にウィンドウ配置を変えたい方は「Switch」をクリックしてください。）



ブレークポイントポイントに止まる

- デバッグの実行が開始され、ブレークポイントを張った場所で処理が止まります。

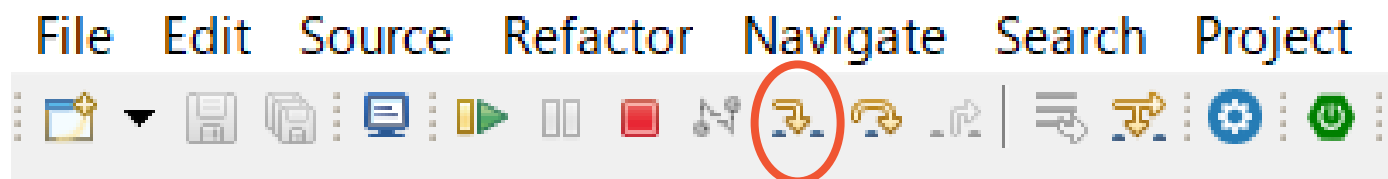
```

3 public class DebugPractice {
4
5     public static void main(String[] args) {
6
7         String s1 = getBanana();
8         System.out.println(s1 + "が食べたい");
9
10        String s2 = getStrawberry();
11        System.out.println(s2 + "が食べたい");
12    }
13
14    public static String getBanana() {
15        return "バナナ";
16    }
17
18
19    public static String getStrawberry() {
20        return "イチゴ";
21    }
22 }

```


ステップイン実行（F5キー）

- ステップインボタン（又はF5キー）を押下すると指定行の関数の中へ処理が移動する



- 今回の例では、getBananaの中に移動します。

```
public static void main(String[] args) {

    String s1 = getBanana();
    System.out.println(s1 + "が食べたい");

    String s2 = getStrawberry();
    System.out.println(s2 + "が食べたい");
}

public static String getBanana() {
    return "バナナ";
}

public static String getStrawberry() {
    return "イチゴ";
}
```

ステップリターン実行（F7キー）

- ステップリターンボタン（又はF7キー）を押下するとステップインで入ったメソッドの中から外に出ます。



- 今回の例では、String s1に戻ります。

```
public static void main(String[] args) {
    String s1 = getBanana();
    System.out.println(s1 + "が食べたい");

    String s2 = getStrawberry();
    System.out.println(s2 + "が食べたい");
}

public static String getBanana() {
    return "バナナ";
}

public static String getStrawberry() {
    return "イチゴ";
}
```

再開（F8キー）

- 「再開」ボタン（又はF8キー）を押下すると、次のブレークポイントまで実行されるか、次のブレークポイントがない場合は最後まで処理が実行されます。



- 流れとしては以上ですが、続けて「終了（Ctrl+F2）、ステップオーバー実行（F6キー）」についても解説します。

ステップオーバー（F6キー）

- ブレークポイントが止まっている状態で、ステップオーバー（F6キー）を押下実行すると、そのまま次の行へ移動します。



- ステップイン実行（F5キー）では、getBanana関数の中に移動しましたが、ステップオーバー実行（F6キー）では、8行目に移動しました。

```

3 public class DebugPractice {
4
5     public static void main(String[] args) {
6
7         String s1 = getBanana();
8         System.out.println(s1 + "が食べたい");
9
10        String s2 = getStrawberry();
11        System.out.println(s2 + "が食べたい");
12    }
13
14    public static String getBanana() {
15        return "バナナ";
16    }
17
18
19    public static String getStrawberry() {
20        return "イチゴ";
21    }
22 }

```



```

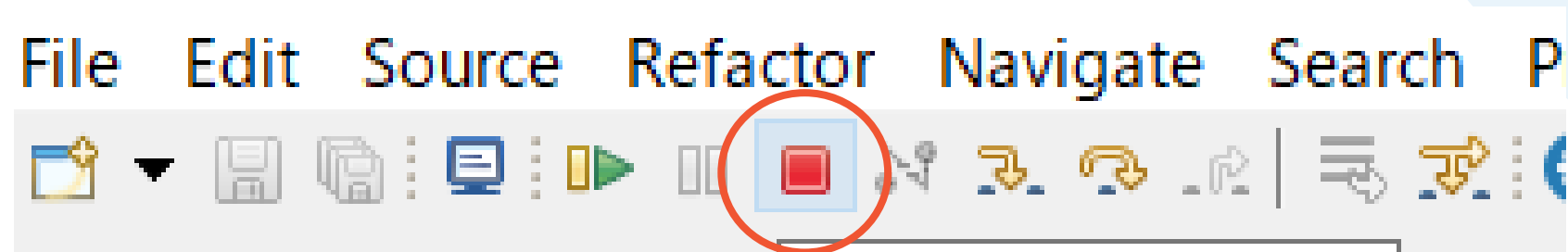
3 public class DebugPractice {
4
5     public static void main(String[] args) {
6
7         String s1 = getBanana();
8         System.out.println(s1 + "が食べたい");
9
10        String s2 = getStrawberry();
11        System.out.println(s2
12    }
13
14    public static String getBanana() {
15        return "バナナ";
16    }
17
18
19    public static String getStrawberry() {
20        return "イチゴ";
21    }
22 }

```

関数の中に行かず、
次の行へ移動

終了 (Ctrl + F2)

- デバッグ実行を終了させたいときは停止ボタン (又は、Ctrl+F2)を押下します。



デバッグの必要性

- なぜデバッグが必要なのか演習問題を解いて考えましょう
- 以下のソースを記述後、デバック操作を行いどのIF文に入るかどうかの確認しなさい。

```

1 public class Ex1 {
2     public static void main(String[] args) {
3
4         String weather = "晴れ";
5
6         if (weather.equals("晴れ")) {
7             System.out.println("洗濯物を外に干します");
8         } else if (weather.equals("曇り")) {
9             System.out.println("乾燥機で乾かします");
10        } else {
11            System.out.println("洗濯物は干しません");
12        }
13    }
14 }

```

デバックの使いどころ

- デバックでは、IF文に入ったかどうか
- 繰り返し文で実行中の変数の値を知る
- 実行中にエラーが発生するソース
⇒どの段階でエラーになっているか

デバックが出来ることで、あらゆる問題に対処することができます。
この場で、デバックをする際の操作方法は必ず取得してください。

練習問題

- 以下のソースを実行し、コンソールに「20」を出力したいが、エラーが発生し、「20」を出力することができない。
- デバック操作を行い、どの時点でエラーが発生しているかを突き止め、コンソールに「20」が出力されるようにソースを修正しなさい。

```

1 public class Ex2 {
2     public static void main(String[] args) {
3         int array2D[][] = new int[][] { { 1, 3 }, { 5, 7, 9 } };
4         int total = 0;
5         for (int i = 0; i < array2D.length; i++) {
6             for (int j = i; i < array2D[i].length; j++) {
7                 total += array2D[i][j];
8             }
9         }
10        System.out.println(total);
11    }
12 }

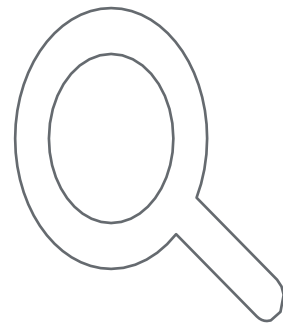
```

デバッグ関係のショートカットキーまとめ

ショートカットキー	内容
Ctrl + Shift + B	ブレークポイントの切り替え
F5	ステップイン実行
F6	ステップオーバー実行
F7	ステップリターン実行
F8	再開
Ctrl + F2	終了



Q&A



まとめ

Sum Up



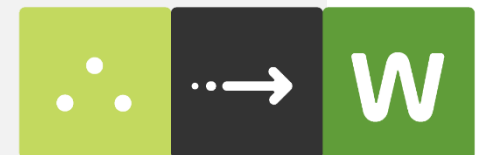
1. EclipseとGitHubの連携方法

2. デバッグ機能の紹介

- ・ ステップイン
- ・ ステップオーバー
- ・ ステップリターン
- ・ 再開
- ・ 終了

Thank you!

From Seeds to Woodland — Shape Your Future.



Shape Your Future