

CS 1103-01 Programming 2 - AY2023-T1

[Dashboard](#) / [My courses](#) / [CS 1103-01 - AY2023-T1](#) / 1 September - 7 September / [Lab 1 Unit 1](#)

Lab 1: Introduction to Eclipse and Javadoc

For those who were not introduced to Eclipse and Javadoc in the CS 1102 course, this lab provides an introduction to programming with the Eclipse IDE (Integrated Development Environment) and Javadoc.

Eclipse is used by the majority of professional Java programmers. As an IDE, it is a unified environment for editing, running, and debugging Java programs. In fact, it is really much more, since plug-ins are available that add support for many other programming languages. Eclipse is a very large and complex system, and it is very likely that you will only ever use a small subset of its features. Fortunately, it is possible to be very productive in Eclipse even if you are familiar with only some of the things that it can do.

Javadoc is a standard syntax for comments in Java source code. Javadoc was introduced in Section 4.5 of the textbook, and you can read about it there.

Programming Exercise: Random Math Quiz

To get you started with Eclipse and to help you get back into programming at the start of a new semester, this lab includes a moderately complex programming assignment. Remember that your program will be graded for style as well as for correctness. It should follow all the rules in the style guide (a link for which is available from the Resources sidebar on the course page), including the use of Javadoc comments (see below).

The assignment is to write a program that administers and grades a math quiz with randomly generated questions of several different kinds. You should target the quiz at an elementary school student who is learning basic arithmetic. The questions should ask the student to perform simple addition, subtraction, multiplication, and division problems. You can include other problems, such as finding remainders, if you want. You should probably stick to integers with a small number of digits.

For each question, the type of problem should be selected at random, and then the numbers that occur in the problem should be randomly generated. (Hint: To get a reasonable division problem, a/b , select b and the answer at random, and compute the value of a .)

The user should have two chances to get the right answer. If the user gets the correct answer to a problem on the first try, they get full credit for the problem. If not, they get a second try. If the user gets the correct answer on the second try, they get half credit. If the user fails to get the correct answer after two tries, tell the user the correct answer.

The quiz consists of ten questions. At the end of the quiz, output a score along with the number of questions that the user got right on the first try and the number of questions that the user got right on the second try.



You should try to write your program in a way that will make it easy to modify and improve. For example, you wouldn't want to hard code the number of questions as a literal constant in your program, would you? The preferred way to write the program is to define classes to represent the concepts that occur in the problem. You will create one such class (along with the main

program) as part of your introduction to Eclipse. It's up to you whether you actually use this class (and possibly improve it) and whether you create additional classes. You should do so only if you are comfortable with the idea -- the most important thing is to have a working program.

This is a non-gradable project, so you do not have to turn in your work.

A perfect program should run correctly; it should provide a good user experience with reasonable math problems and nice, clear I/O; and should follow all the rules of good programming style. An outstanding program must also exhibit a good design that will make it easy to extend the program, for example to use a different number of questions, different types of questions, and possibly different scoring rules.

Running Eclipse



Eclipse is available for Linux, Windows, and Mac OS. (For a Mac, you will need Mac OS 10.4 or higher.)

To run Eclipse, you must first have a Java JDK Version 5.0 or higher working on your computer. If you do not already have it on your computer, you can download it. For Linux, the JDK should be included in your distribution's available software. Windows users can get the JDK 6 from Sun's Java download site,

<http://java.sun.com/javase/downloads/index.jsp>

You need to download "Java SE Development Kit (JDK) 6"; a link to installation instructions can be found on the same web page. Recent versions of Mac OS already include Java; version 6 is not yet available for Mac OS, but version 5 will work fine for this course.

Once you have Java working, you can download the most recent version of Eclipse from the Eclipse downloads page, <http://www.eclipse.org/downloads/>. You can download the "Eclipse IDE for Java Developers"; you won't need the much larger "Eclipse IDE for Java EE Developers." Note that the latest version of Eclipse will be a little different from the version that you will use in the lab. For example, the newer version will want to put your source code files in a directory named "src." However, the differences are minor and should not be a problem.

Eclipse organizes your programming projects into "workspaces." Each workspace corresponds to a directory. These workspace directories hold all the files that are used by your projects, including Java source code files and compiled class files. Usually, you should let Eclipse have complete control over the workspace directories, and you should not directly change any files that are stored in them. It is possible to have as many workspaces as you want, but for this course, you will probably keep all of your projects in a single workspace.

When Eclipse starts up for the first time, it asks you to select a workspace. By default, the name of the workspace will be "workspace", but a good habit is to change it to "eclipse-workspace". There is a box that you can check if you don't want to be asked to select a workspace each time you run Eclipse.

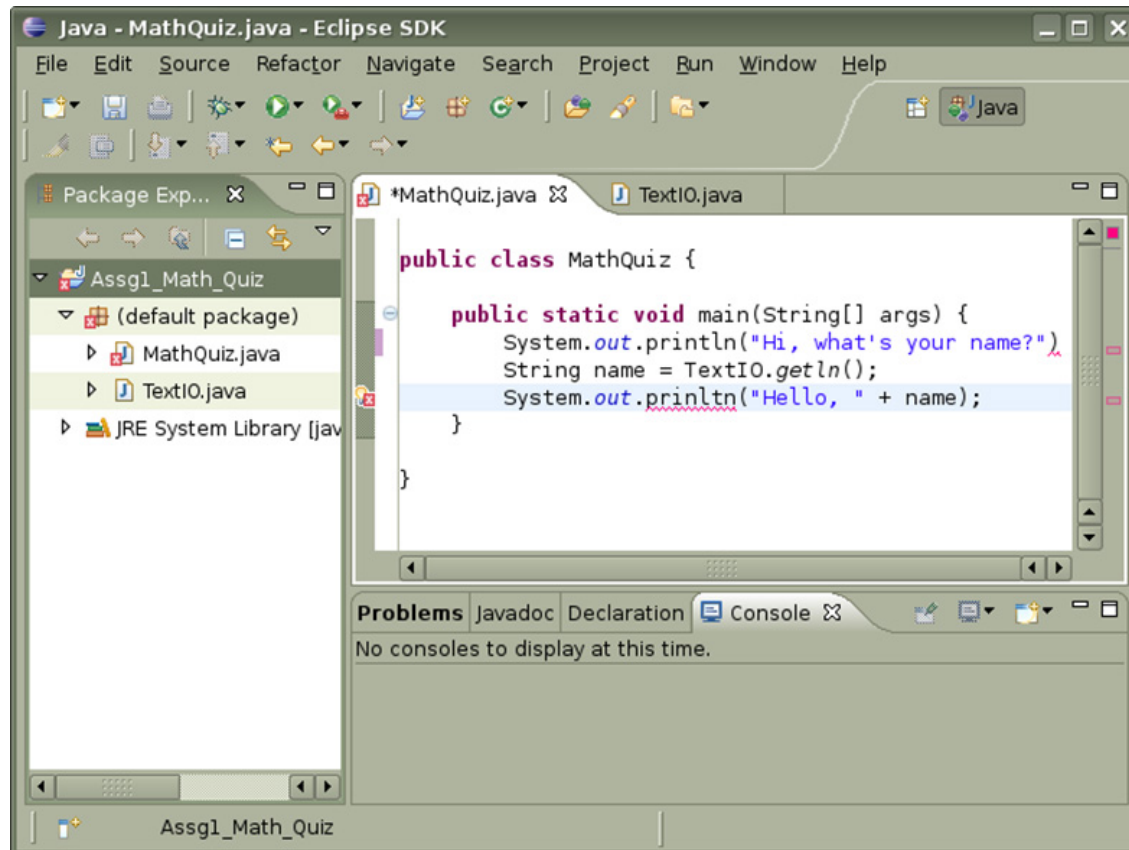
The first time that you run Eclipse, its window will be filled with a **Welcome** screen. The icons on this screen link to a large amount of information about Eclipse. You can browse through this information some time if you like, but for now, just close the **Welcome** screen by clicking the "X" next to the word **Welcome** near the top-right corner of the window. If you want to get back to the **Welcome** screen, just select **Welcome** from the **Help** menu.



Eclipse uses the terms view and perspective to describe the way information is organized and presented in its window. The window typically contains several views. Each view contains a certain type of information. All the views that are visible in the

window constitute a perspective. A perspective is meant to contain all the views that are used in some particular activity. For now, you will just use the Java perspective, which is used for Java programming. Later, you will learn about the Debug perspective, which is for debugging programs. Each perspective is highly customizable. You can add and delete views and move them around. *If you ever delete a view accidentally, you can get it back by selecting it from the **Show View** submenu in the **Window** menu.*

After you have closed the **Welcome** screen, the window will show the Java perspective. Initially, all the views are empty. Here is what the Java perspective might look like after a little work has been done:



In this window, several views that are not often used have been closed. Such views include **Outline** and **Hierarchy**. Remember that a view can be closed by clicking the small "X" next to the name of the view, and it can be reopened using the **Window/Show View** menu.

The **Package Explorer** view on the left is central to much of the work that you do in Eclipse. It contains a list of your programming projects and the Java files and resources that are contained in those projects. In the image above, there is just one project, **Assg1_Math_Quiz**. Clicking on the small triangle next to the project name will show/hide the resources contained in the project.

The lower right section of the window contains several views. Currently, the **Console** view is showing. To see one of the other views, such as **Problems** or **Javadoc**, just click on the tab that contains the name of the view. Sometimes, another view will pop up automatically in this area to show the output of some command. When you run a program, standard input and output are shown in the **Console** view. Errors and warnings from the Java compiler are displayed in the **Problems** view.

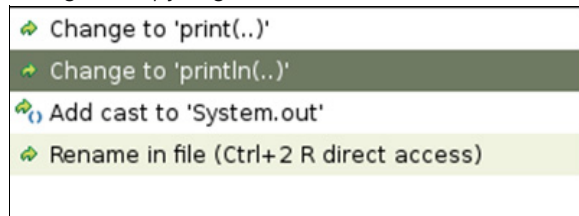
The central area of the window is occupied by editor views. Here, two Java files are open for editing: **MathQuiz.java** and **TextIO.java**. Currently, **MathQuiz.java** is showing; to see **TextIO.java** instead, click its name.

The view of **MathQuiz.java** shows several of the nifty features of the Java editor. The source code that you type is checked for syntax errors as you type. Errors are marked with small red carets at the bottom of the line, like the one at the end of the first line in the **main()** routine. The error is also marked by a red rectangle in the right margin of the editor; if you hover your mouse



over the red rectangle, you see the error message for the error. In this case, you are told that a semicolon is missing. In the third line of `main()`, the word `"println"` is underlined with error markers. (It's a misspelling of `"println"`.) This error has an error light bulb (🔧) in the left margin of the editor.

The light bulb means that Eclipse is not just telling you about the error but is offering you some ideas for fixing it. If you click the light bulb, you get a list of actions that can be taken to fix the problem. For the above picture, the list is:



Double-clicking on an action, such as `"Change to println()"`, will apply that option automatically. Sometimes, you will see a warning light bulb (⚠️). A warning indicates a potential problem, but not an error. Warnings will not prevent the source code from being compiled successfully.

In fact, Eclipse might be a little too enthusiastic in marking warnings and errors. You do not necessarily have to fix every warning. And you do not have to fix every error as soon as it appears.

In fact, it's impossible to do so, and in some cases the error will go away by itself after you've typed in more of your program. And remember that the fix for a programming error does not always go at the location of the error; sometimes the problem is elsewhere in the file. Furthermore, Eclipse's error system is only effective if you routinely get most of your program correct in the first place -- don't expect Eclipse to make solid Java programming skills unnecessary!


Your First Project

It's time for you to start using Eclipse. Startup Eclipse, as described above. Close the **Welcome** screen (and maybe the **Outline** view as well).

To create your first project, right-click in the **Project Explorer**. In the pop-up menu that appears, go to the **New** submenu, and select Java Project. A **New Java Project** wizard pops up. All you have to do is enter a name for the project, in the box labeled Project Name. Enter `"Math Quiz"`, or `"Lab1"`, or any other name that you like. (Warning: Don't use funny characters such as `"&"` or `":"` in the name of the project, since they can cause problems when you work with the files outside Eclipse. The project name will be used as the name of a directory inside your workspace directory.) After entering the project name, click the **Finish** button. The project will be added to **Project Explorer**, and a directory of the same name will be created in your Eclipse workspace directory.

Now, you need to create a new Java class file in your project. To do this, right-click the project name in the **"Package Explorer"** view. In the pop-up menu, go to the **New** submenu again, and select **Class**. A class creation dialog box will pop up. Again, you just have to fill in the name of the class, and click the **Finish** button. Note that you have to enter the name of the **class**, not the name of the **file**, so don't try to add a `.java` extension to the name. The name must be a valid Java class name. Use `"MathQuiz"`, or whatever you like. The class will be added to the package in the **Package Explorer** view, and the Java file will be created in the project directory. Furthermore, a Java editor will open, showing you the initial contents of the file. As you can see, Eclipse has already added the declaration of the class. All you have to do is fill it in! Note that you can close the editor in the usual way; to open the file again, double-click its name in the **Project Explorer**.

As you can see in the **Package Explorer**, the class is added to the `"default package."` This just means that the class is not declared to belong to any package. Later in the term, you will be working with packages, but for now the default package is OK, even though Eclipse will display a warning that `"Use of the default package is discouraged."`

You will probably want to use the `TextIO` class in the Random Math Quiz project. To make it available to your program, you have to add it to the project. The easiest way to get this file into your project is to copy-and-paste it from a directory window:  a directory window for the `source` directory in this unit that contains `TextIO.java`. Right-click its file icon and select **Copy** from the pop-up menu. Then right-click the project name in Eclipse's **Project Explorer** and select **Paste** from the pop-up menu. `TextIO` should appear in the default package.

Eclipse often has alternative ways of doing things. Another way to create projects and classes is with the "New" submenu in the "File" menu. Even easier is to use the "create" buttons in the toolbar. These are a group of three small buttons at the top of the Eclipse window:




Click on the left button in this group to create a new project. Click the right button to create a new class. The middle button is for creating packages. (Note that when you create a class using the button, you should first click in the **Package Explorer** to select the location of the class that you are creating. Otherwise, you'll have to enter the location by hand in the class creation dialog box.)

Now, to create and run a program. Add the following main() routine, or something similar, to the class that you created above:

```
public static void main(String[] args) {  
    System.out.print("What's your name? ");  
    String name = TextIO.getln();  
    System.out.println("Pleased to meet you, " + name);  
}
```

The program is compiled automatically. To run it, right-click either on the name of the class in the **Project Explorer** or on the editor window that contains the program. In the pop-up menu, go to the **Run As** submenu, and select **Java Application** from the submenu. The program will start. The output from the first print statement appears in the **Console** view in the bottom right area of the Eclipse window. In order to type a response, you must **first click the Console window!** Type in your response and press return. The last line of the program will be executed, and the program will end.

After running the program once, you can run it again simply by clicking on the **Run** button () in the toolbar. Clicking this button will re-run the program that was run most recently. If you click the little black arrow to the right of the button, you'll get a list of all the programs that you have run; select a program from this list to run it.

Before starting serious work on the programming assignment, create one more test class. Create a class named **AdditionProblem** with the following definition:

```
public class AdditionProblem {  
  
    private int x,y,answer;  
  
    public AdditionProblem() {  
        x = (int)(10 + 40*Math.random());  
        y = (int)(30 * Math.random());  
        answer = x + y;  
    }  
  
    public String getProblem() {  
        return "Compute the sum: " + x + " + " + y;  
    }  
  
    public int getAnswer() {  
        return answer;  
    }  
}
```

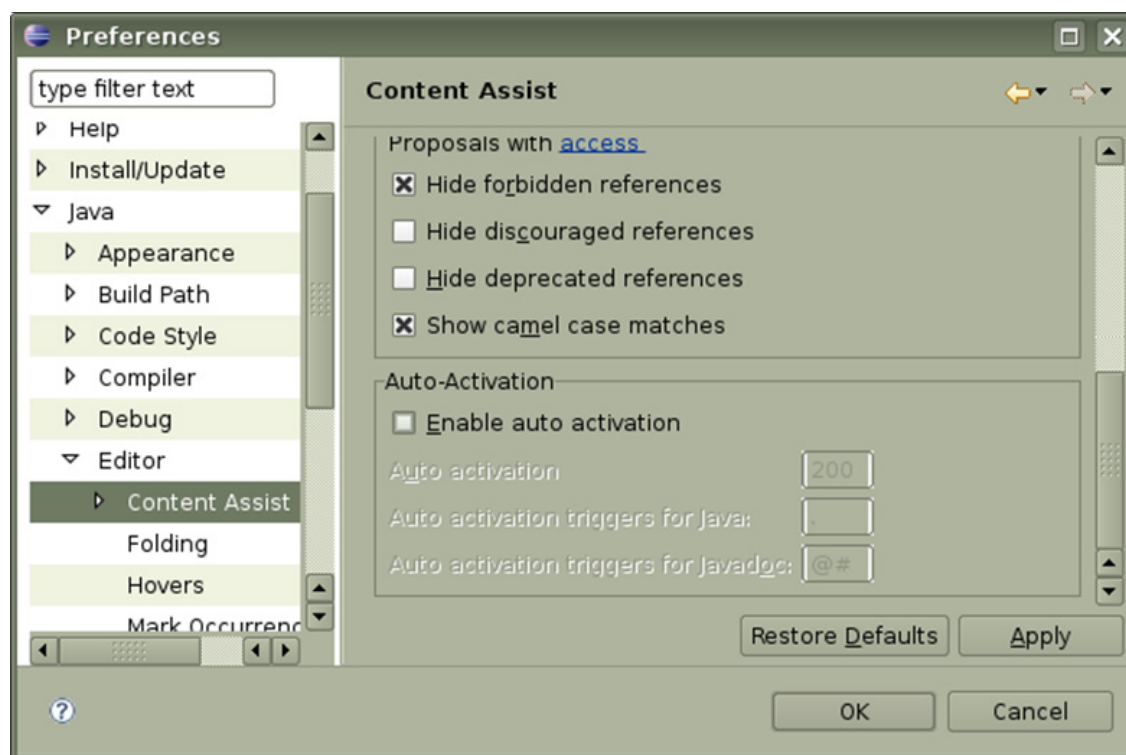


```
}
```

Now, change the `main()` routine so that it creates an **AdditionProblem**, displays to the user the problem returned by the `getProblem()` method, reads an answer from the user, and checks to see whether the answer entered by the user agrees with the correct answer as returned by `getAnswer()`. **(It's a good idea to complete this little exercise if you want to use classes like this one in your real math quiz program; doing the exercise will make sure that you know how to use the class.)**

You have surely already noticed that the Java editor in Eclipse does a certain amount of work for you. It automatically inserts indentation. When you type a "{" and press return, the matching "}" is inserted in the correct position. Furthermore, when you type a period, a list of things that can follow the period will pop up after a short delay, and you can select an item from the list instead of typing it in. This is called Content Assist. You can invoke Content Assist at any time while you are typing by pressing Control-Space. If you do this in the middle of a variable name or method name, pressing Control-Space will either complete the name or offer a list of possible completions. It can be used in other ways as well. For example, if you press Control-Space after typing the "(" at the beginning of a method's parameter list, you will get information about the parameters of the method. By the way, when Content Assist pops up a list, you can get rid of the list by pressing the Escape key.

Content Assist is a good thing, but some find the way it pops up automatically while typing to be very annoying. You can turn off this feature in the Eclipse **Preferences**. Select the Preferences command from the **Window** menu. In the **Preferences** dialog box, click the little plus sign next to **Java** to open the list of Java preferences (if necessary), then click the little plus sign next to **Editor**, and finally click Content Assist. In the **Content Assist** preferences, uncheck the box labeled **Enable Auto Activation** and click **OK**. It looks like this:



Eclipse has a huge number of preference settings that can be used to customize the environment. Most of the default settings are OK, but there are a few that you should change:

- Under **Java > Editor > Mark Occurrences**, unselect **Keep marks when the selection changes**.
- Under **Java > Compiler > Errors/Warnings > Potential Programming Problems**
 - Change **Serializable class without serialVersionUID** from Warning to Ignore
 - Change "Possible accidental boolean assignment", "'switch' case fall-through" and "Null reference" from Ignore to Warning.



Eclipse has a lot of other useful features. We will encounter more of them as time goes on, and you can undoubtedly discover a few new ones on your own. But here are a few of the most useful:

- Eclipse can fix the indentation of your source code. Just highlight a segment of code and press Control-I. Note that fixing the indentation can help you find mismatched braces and incorrectly nested blocks of code in your program. **(With this feature available, there is no excuse for badly indented programs.)**
- Highlight a segment of code and hit Control-/ to "comment out" that code by adding `/**` to the beginning of each line. If the code was already commented out, it will be commented back in, by removing the `/**`'s.
- In an editor, if you double click just after a `"{"` or `"}"`, the entire block will be highlighted, making it easier to figure out just where the block begins or ends.
- Highlight the name of a class, method, or variable and hit F3. You will be taken to the declaration of the highlighted name, even if it is in another file.
- When an uncaught exception occurs in your program, the stack trace of the exception appears in the Console view. The stack trace contains links to lines in the program, and clicking on the link will take you to that line in an editor view.
- Eclipse can do smart renaming. When you rename a variable, for example, all references to that variable will be changed to use the new name. To do a smart rename, just highlight the name where it is declared, and select the **Rename** command from the **Refactor** menu. (Refactoring is a general term that refers to rearranging or modifying code to improve it or make it more general.) The same thing works for renaming methods and classes.
- The **Source** menu offers several options for automatically generating code. I probably use **Generate Getters and Setters** most often, but the commands for overriding methods and for generating constructors are also useful.
- When you modify a section of code, a thin gray bar appears in the left margin of the editor next to the code that you have modified. Right-click that bar to get the option of reverting the code segment back to what it was before you started editing. When you delete some lines, a tiny black rectangle appears in the left margin. Right-click near this rectangle for the option of restoring the deleted lines. (The bar and rectangle only last until you save the file, but Eclipse keeps several versions of each file in a "history", and you can revert to earlier versions from this history if you need to.)

Javadoc

Javadoc comments can be processed to produce documentation of your code's API in the form of web pages. The API documentation for Java's standard classes was created with Javadoc. See it here: <http://java.sun.com/javase/6/docs/api/index.html>.

Javadoc is thoroughly integrated into Eclipse. If you highlight the name of a class, method, or variable, its Javadoc documentation (if any) appears in the Javadoc view. If you just hover your mouse over a name, the **Javadoc** documentation appears in a pop-up window. When Content Assist shows a list of variable or method names, it will also show Javadoc documentation for items on the list. This makes it very worthwhile to use Javadoc comments in your code, even if you don't plan to generate web pages from the comments.

Furthermore, Eclipse will generate the outline of a Javadoc comment for you. To have it do this, just position the cursor on the line before a declaration, type `/**` (the first three characters of a Javadoc comment), and press return. An outline of the comment, including any appropriate Javadoc tags (such as `@author`, `@param`, `@return`, and `@throws`) will appear. Another way to add a comment is to hit Shift-Alt-J while typing in an editor window. A comment will be added to the item that contains the cursor.



Some things to keep in mind: A Javadoc comment always comes *before* the declaration that is being commented on. Any

Javadoc tags, such as @param and @return, must come at the end of the comment, after any other text in the comment. Javadoc comments can contain HTML markup tags such as <p> or This means that you should not use the characters "&" or "<" in a Javadoc comment; Write them as "&" or "<" instead.

In this course, you are **required** to use Javadoc comments for any non-private classes, member variables, and methods that you write. This requirement starts now, with the Random Math Quiz programming assignment.

Last modified: Thursday, 24 June 2021, 8:19 AM

UoPeople Clock (GMT-5)

All activities close on Wednesdays at 11:55 PM, except for Learning Journals/Portfolios which close on Thursdays at 11:55 PM **always following the clock at the top of the page.**

Due dates/times *displayed* in activities will vary with your chosen time zone, however you are still bound to the **11:55 PM GMT-5** deadline.

◀ Self-Quiz Unit 1


Jump to...

Lab 2 Unit 1 ▶

[Disclaimer Regarding Use of Course Material](#) - [Terms of Use](#)

University of the People is a 501(c)(3) not for profit organization. Contributions are tax deductible to the extent permitted by law.
Copyright © University of the People 2021. All rights reserved.

You are logged in as [Abel Lifaefi Mbula](#) ([Log out](#))

 www.uopeople.edu



Resources

[UoPeople Library](#)

[Orientation Videos](#)

[LRC](#)

[Syllabus Repository](#)

[Career Resource Center](#)

[Honors Lists](#)

Links

[About Us](#)

[Policies](#)

[University Catalog](#)

[Support](#)

[Student Portal](#)

Instructors

[Instructor Portal](#)

[CTE Center for Teaching Excellence](#)

[Office 365](#)

[Tipalti](#)

[Contact us](#)

[English \(en\)](#)

[English \(en\)](#)

[العربية \(ar\)](#)



[Data retention summary](#)

[Get the mobile app](#)

