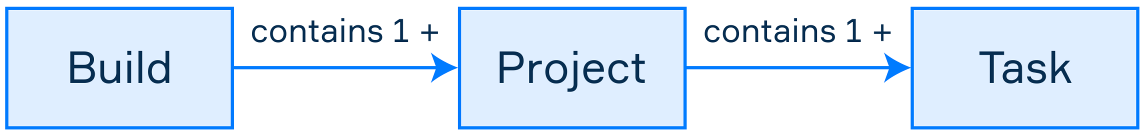
Basic project with Gradle

The key concepts of Gradle

Let's start with an introduction to the key concepts in
Gradle: **projects** and **tasks**.

- A **project** might represent either **something to be built** (e.g. a JAR file or ZIP archive) or **a thing to do** (e.g. deploying the application). Every Gradle build contains one or more projects.
- A **task** is a single piece of work that a build performs. This can include compiling classes, running tests, generating docs, and so on. Every project is essentially a collection of one or several tasks.

The following picture illustrates the relationships between these concepts:



In simple cases, a build will contain only a single project with several tasks.

Initializing a basic project managed by Gradle

Let's initialize a new project with Gradle using a terminal in your OS.

In the future, you will most likely not have to do this manually since modern

IDEs can do this for you automatically.

1. Create a new directory to store files of your project and go to it.

```
mkdir gradle-demo
cd gradle-demo
```

Invoke the gradle init command to generate a simple project.
Modern versions of Gradle will ask you to fill several parameters in a dialogue form.
To get acquainted with the process just choose basic as the type of project and Groovy as the build script DSL.
This command will produce the following output:

```
> Task :init

BUILD SUCCESSFUL in 10s
2 actionable tasks: 2 executed
```

Gradle performed some tasks for you and now there is a simple project with the most basic structure:

```
.
├── build.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
└── settings.gradle
```

Here is brief info about all the generated files:

- The build.gradle file is a primary file that specifies the Gradle's project, including its tasks and external libraries. Here, it is located in the gradle-demo folder and can be created after invoking the gradle init command. For now, this file doesn't contain anything useful, but in real projects it is often updated with new information.

- The files gradle-wrapper.jar, gradle-wrapper.properties, gradlew and gradlew.bat belong to Gradle Wrapper which allows you to run Gradle without its manual installation.
- The settings.gradle file specifies which projects to include in your build. This file is optional for a build that has only one project, but it is mandatory for a multi-project build.

Let's build our project invoking the gradle build command from the same location where build.gradle resides. It will produce an output like this:

```
> Task :buildEnvironment
```

```
------------------------------------------------------------
Root project
------------------------------------------------------------

...

BUILD SUCCESSFUL in 725ms
1 actionable task: 1 executed
```
So, the project was successfully built with one executed task.

You can also invoke build and other commands like ./gradlew build for Unix-based systems and gradlew.bat build for Windows. It will automatically download Gradle and run the specified command. Using wrappers allows developers to start working with a Gradle-based project without having to install it manually.

Creating a basic project via Intellij IDEA

You can also create a new gradle project via Intellij IDEA. For this, choose New Project option on the starting screen or click **File > New > Project...** in the main menu.

In the New Project window, select **New Project** option and then choose the desired language (Java or Kotlin) and the build system (Gradle).

Here you can also choose the project name and location and specify some additional options, such as the JDK version or Gradle DSL language, etc. When you are done, click **Create** button.

In a few moments, a new Gradle managed project will be created and built:

It has the same structure as if it was built using terminal commands.

Modifying the build file

Let's make our build more interesting by adding some properties and one task to the build.gradle file using Groovy DSL.
```
description = "A basic Gradle project"

task helloGradle {
   doLast {
      println 'Hello, Gradle!'
   }
}
```
Here, we set the description property and define a simple task that prints a 'hello' message. There is an output after executing the task with the gradle -q

```
helloGradle command:
> Task :buildEnvironment

------------------------------------------------------------
Root project - A basic Gradle project
------------------------------------------------------------

...

> Task :helloGradle
Hello, Gradle!

BUILD SUCCESSFUL in 831ms
2 actionable tasks: 2 executed
```

This build was completed with 2 tasks executed. Our new task printed the Hello, Gradle! message. In addition, we modified the description of the project in the build. The -q argument just simplifies the command output.

You can also use Kotlin as DSL inside the build file. To allow it, you need to specify Kotlin as DSL when creating a project. In this case, the name of the file will be build.gradle.kts.

The list of all the tasks
If you would like to see all the possible Gradle tasks to perform, just run the gradle tasks --all command. The list will include our tasks as well:

```
> Task :tasks

------------------------------------------------------------
Tasks runnable from root project - A basic Gradle project
------------------------------------------------------------

Build Setup tasks
-----------------
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Help tasks
----------
buildEnvironment - Displays all buildscript dependencies declared in root
project 'gradle-demo'.
...

Other tasks
-----------
helloGradle
```

In a real project, the list of tasks will be much larger, because, in addition to standard tasks, it will contain a lot of tasks from various plugins (like Java or Kotlin plugin).

We've considered all Gradle-related files from the generated simple project in isolation from any source code files.