1. What is polymorphism?

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

In [programming language theory](https://en.wikipedia.org/wiki/Programming_language_theory) and [type theory](https://en.wikipedia.org/wiki/Type_theory), polymorphism is the provision of a single [interface](https://en.wikipedia.org/wiki/Interface_(computing)) to entities of different [types](https://en.wikipedia.org/wiki/Data_type) or the use of a single symbol to represent multiple different types.The concept is borrowed from a principle in biology where an organism or species can have many different forms or stages.

In simple terms, polymorphism is a single action that can be performed in different ways.

2. What is the difference between method overloading and method overriding?

When coming down to the difference between method overloading and method overriding, the key differences are:

1. Method overloading is a compile-time polymorphism on the other side Method overriding is a run-time polymorphism.
2. Method overloading occurs within the class, and Method overriding s performed in two classes with inheritance relationships.
3. In method overloading, methods must have the same name and different signatures, and In method overriding, methods must have the same name and same signature.
4. In method overloading, the return type can or can not be the same, but we have to change the parameter, and In method overriding, the return type must be the same or co-variant.
5. Private and final methods can be overloaded but private and final methods can’t be overridden.
6. Method overloading is a compile-time polymorphism and method overriding is a run-time polymorphism.

3. What is an abstract class, and how can you recognize an abstract class in Java?

An abstract class is a class that is declared as an abstract class and can not be instantiated but can be subclassed.

It also may or not may have an abstract method. The method will not have a body.

To recognize an abstract class, an abstract class will have the declaration of an abstract class and it will have at least one abstract method.

4. Explain phases of the maven build lifecycle.

Maven is based on the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

here are three built-in build lifecycles: default, clean, and site. The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, and the site lifecycle handles the creation of your project's website.

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the [Lifecycle Reference](https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference)):

A build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in the life cycle. As an example, a typical Maven Build Lifecycle consists of the following sequence of phases.

* validate - validate the project is correct and all necessary information is available
* compile - compile the source code of the project
* test - test the compiled source code using a suitable unit testing framework. These tests should not require the code to be packaged or deployed
* package - take the compiled code and package it in its distributable format, such as a JAR.
* verify - run any checks on results of integration tests to ensure quality criteria are met
* install - install the package into the local repository, for use as a dependency in other projects locally
* deploy - done in the built environment, copies the final package to the remote repository for sharing with other developers and projects.

5. What is POM file?

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples of this are the build directory, which is the target; the source directory, which is src/main/java; the test source directory, which is src/test/java; and so on. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

Some of the configurations that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists, and such can also be specified.

POM stands for Project Object Model. It is the fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.

The POM contains information about the project and various configuration detail used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal. Some of the configurations that can be specified in the POM are the following −

* project dependencies
* plugins
* goals
* build profiles
* project version
* developers
* mailing list

Before creating a POM, we should first decide the project group (groupId), its name (artifactId), and its version as these attributes help in uniquely identifying the project in the repository.

6. What is the purpose of locators?

As quality engineers, we all know that for testing a web-based application, we need a perform specific actions(such as click, type, etc.) on the HTML elements. Now, when we go for automation of those applications, the automation tool should also be capable of performing the same operations on the HTML elements that a human is capable of. So, now the question comes, how do these automation tools identify which HTML element they need to perform a particular operation? The answer to this is "Locators in Selenium".

Locators are the way to identify an HTML element on a web page, and almost all UI automation tools provide the capability to use locators for the identification of HTML elements on a web page. Following the same trend, Selenium also possesses the ability to use "Locators" for the identification of HTML elements and is popularly known as "Selenium Locators". Selenium supports various kinds of locators.

7. What are the different types of locators commonly used in selenium?

Selenium supports the following locators:

* ClassName – A ClassName operator uses a class attribute to identify an object.
* CSS selectors – CSS is used to create style rules for webpages and can be used to identify any web element.
* Id – Similar to class, we can also identify elements by using the ‘id’ attribute.
* link text – Text used in hyperlinks can also locate element
* name – The name attribute can also identify an element
* partial link text – Part of the text in the link can also identify an element
* tagName – We can also use a tag to locate elements
* Path – Xpath is the language used to query the XML document. The same can uniquely identify the web element on any page.

8. What is the difference between Abstract class and interface?

A key difference is that classes can implement more than one interface, but they can extend only one abstract class. This is a design decision based on the fact that multiple inheritances (extending more than one class) can cause code deadlocks. Java's engineers decided to avoid that.

Another difference is that interfaces can be implemented by classes or extended by interfaces, but classes can be only extended.

From an object-oriented programming perspective, the main difference between an interface and an abstract class is that an interface cannot have a state, whereas the abstract class can have state with instance variables.

9. Create an abstract class Animal with the following abstract methods:

* makeSound()
* canEat()

Another method aboutMe() which displays message "I am Animal"

Create two concrete classes Cat and Parrot which extends Animal class. Cat class has method favouriteToy() which prints the favorite toy of cat. Dog class has a method canRun() which display message "dog can run" .