``````````JAVA Interview Questions:

As an automation tester with 2 years of experience, you'll be expected to have a solid grasp of Java fundamentals, object-oriented programming (OOP) concepts, collections, exception handling, and how to apply these within the context of automation frameworks like Selenium and TestNG. You should also be familiar with build tools like Maven and version control systems like Git.

Here's a comprehensive list of commonly asked interview questions, categorized for easier preparation:

**Core Java Concepts**

1. **What are the key features of Java?**
   * *Expected Answer:* Platform Independent, Object-Oriented, Simple, Secure, Multithreaded, High Performance, Distributed, Dynamic.
2. **Explain the difference between JDK, JRE, and JVM.**
   * *Expected Answer:*
     + **JVM (Java Virtual Machine):** The runtime environment that executes Java bytecode. It's the core of Java's platform independence.
     + **JRE (Java Runtime Environment):** Includes the JVM and the necessary class libraries and other support files to run Java applications. It doesn't include development tools.
     + **JDK (Java Development Kit):** Includes JRE along with development tools like the compiler (javac), debugger, etc. It's used for developing Java applications.
3. **What are data types in Java? Explain primitive and non-primitive data types.**
   * *Expected Answer:*
     + **Primitive:** byte, short, int, long, float, double, boolean, char. They store direct values.
     + **Non-Primitive (Reference Types):** String, Arrays, Classes, Interfaces. They store references to objects.
4. **Explain the main method in Java (public static void main(String[] args)).**
   * *Expected Answer:*
     + public: Accessible from anywhere.
     + static: Can be called without creating an object of the class.
     + void: Doesn't return any value.
     + main: The entry point for JVM to start program execution.
     + String[] args: Used to pass command-line arguments to the program.
5. **What is a constructor? Differentiate between a default constructor and a parameterized constructor.**
   * *Expected Answer:*
     + A constructor is a special method used to initialize objects. It has the same name as the class and no return type.
     + **Default Constructor:** Provided automatically by Java if no constructor is explicitly defined. Initializes member variables with default values.
     + **Parameterized Constructor:** Takes arguments to initialize member variables with specific values.
6. **What is the difference between == and .equals() in Java?**
   * *Expected Answer:*
     + ==: Compares object references (memory addresses). For primitive types, it compares values.
     + .equals(): Compares the content/value of objects. It's a method from the Object class that can be overridden to provide custom comparison logic.
   * **When to use which:**
   * **Use == when:**
   * Comparing two primitive values.
   * You want to check if two object references point to the *exact same object* in memory.
   * **Use .equals() when:**
   * You want to compare the *contents* or *logical equivalence* of two objects.
   * You are dealing with objects of classes that have meaningfully overridden the equals() method (like String, Integer, Date, etc., or your own custom classes).
7. **Explain this and super keywords.**
   * *Expected Answer:*
     + this: Refers to the current object. Used to distinguish instance variables from local variables if they have the same name, or to call other constructors of the same class.
     + super: Refers to the immediate parent class object. Used to call parent class constructors or methods.

**Object-Oriented Programming (OOP) Concepts**

1. **What are the four pillars of OOP? Explain each with an example relevant to test automation.**

Go to the bottom of the page for answers with detailed explanation..

1. **What is an Interface in Java? When would you use it in automation?**
   * *Expected Answer:* A blueprint of a class. It can only contain abstract methods and static final variables (constants). A class can implement multiple interfaces. Used to achieve abstraction and multiple inheritance (of behavior).
   * *Ex:* In Selenium Webdriver is an interface and it can only have abstract methods(methods without method body).
   * *Automation Use Case:* Defining common actions or contracts across different browser types or test components, e.g., an IBrowser interface with methods like open(), close(), findElement(), etc.
2. **What is an Abstract Class? Differentiate it from an Interface.**
   * *Expected Answer:*
     + **Abstract Class:** Can have abstract and non-abstract methods, constructors, and instance variables. A class can extend only one abstract class. Cannot be instantiated directly.
     + **Differences:**
       - Interface methods are implicitly public and abstract (before Java 8). Abstract classes can have concrete methods.
       - A class can implement multiple interfaces but can extend only one abstract class.
       - Interfaces have final and static variables by default. Abstract classes can have non-final and non-static variables.

**Collections in Java**

1. **What is the Java Collections Framework?**
   * *Expected Answer:* A set of interfaces and classes that represent groups of objects as a single unit. Provides a unified architecture for storing and manipulating collections of objects.
2. **Explain the hierarchy of the Collections Framework.**
   * *Expected Answer:* Iterable -> Collection -> List, Set, Queue. Map is a separate interface but considered part of the framework.
3. **Differentiate between ArrayList and LinkedList. When would you use each in automation?**
   * *Expected Answer:*
     + **ArrayList:** Implements List interface using a dynamic array. Good for fast random access (using index). Slower for insertions/deletions in the middle.
     + **LinkedList:** Implements List and Deque interfaces using a doubly linked list. Good for fast insertions/deletions. Slower for random access.
     + *Automation Use Case:* ArrayList for storing a list of web elements where you might access them by index. LinkedList less common but could be used if you frequently add/remove elements from the beginning/end.
4. **Differentiate between HashSet and HashMap. When would you use each in automation?**
   * *Expected Answer:*
     + **HashSet:** Implements Set interface. Stores unique elements; order is not guaranteed. Uses hashing for storage.
     + **HashMap:** Implements Map interface. Stores key-value pairs; keys must be unique. Order is not guaranteed.
     + *Automation Use Case:* HashSet to store a list of unique strings (e.g., unique product IDs from a page). HashMap to store test data (e.g., HashMap<String, String> userData = new HashMap<>(); userData.put("username", "testuser");).
5. **Explain the difference between Iterator and ListIterator.**
   * *Expected Answer:*
     + **Iterator:** Universal iterator. Can traverse forward only. Can remove elements.
     + **ListIterator:** Specific to List implementations. Can traverse forward and backward. Can add, set, and remove elements.

**Exception Handling**

1. **What is exception handling in Java? Why is it important in automation testing?**
   * *Expected Answer:* A mechanism to handle runtime errors gracefully so that the normal flow of the application can be maintained.
   * *Importance in Automation:* Prevents test script crashes due to unexpected element not found, network issues, etc. Allows for robust test execution, reporting failures, and taking screenshots on error.
2. **Explain try, catch, finally, throw, and throws keywords.**
   * *Expected Answer:*
     + try: Block of code where exceptions might occur.
     + catch: Block of code to handle a specific type of exception thrown from the try block.
     + finally: Block of code that always executes, regardless of whether an exception occurred or was caught. Used for cleanup.
     + throw: Used to explicitly throw an exception.
     + throws: Used in a method signature to declare that a method might throw a certain type of checked exception.
3. **Differentiate between Checked and Unchecked Exceptions.**
   * *Expected Answer:*
     + **Checked Exceptions:** Checked at compile-time. The compiler forces you to handle them (using try-catch or throws). Examples: IOException, SQLException, FileNotFoundException.
     + **Unchecked Exceptions (Runtime Exceptions):** Not checked at compile-time. Occur at runtime. Programmers are not forced to handle them (though it's good practice for some). Examples: NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException.
4. **How do you handle StaleElementReferenceException in Selenium?**
   * *Expected Answer:* This occurs when a web element is no longer attached to the DOM. Common ways to handle it include:
     + Re-finding the element: Locating the element again just before interacting with it.
     + Using explicit waits: Waiting for the element to be clickable or visible before interaction.
     + Refreshing the page (less common but can be an option).

**Selenium WebDriver**

1. **What is Selenium WebDriver? What are its advantages?**
   * *Expected Answer:* An open-source API for automating web browsers. It provides a programming interface to interact with web elements directly.
   * *Advantages:* Supports multiple browsers, multiple programming languages (Java, Python, C#, etc.), open-source, robust API for element interaction.
2. **Explain the architecture of Selenium WebDriver.**
   * *Expected Answer:* Client Libraries (your Java code) -> JSON Wire Protocol (sends commands over HTTP) -> Browser Drivers (Chrome Driver, Gecko Driver, etc.) -> Real Browsers.
3. **What are different types of locators in Selenium? Which one do you prefer and why?**
   * *Expected Answer:* ID, Name, ClassName, TagName, LinkText, PartialLinkText, CSS Selector, XPath.
   * *Preference:* ID is preferred if available (fastest and most reliable). CSS Selectors are generally preferred over XPath due to performance and readability, especially for dynamic elements. XPath is powerful for complex scenarios or when other locators aren't sufficient.
4. **What are implicit and explicit waits in Selenium? When to use which?**
   * *Expected Answer:*
     + **Implicit Wait:** Sets a default waiting time for the driver to find an element before throwing a NoSuchElementException. Applied globally.
     + **Explicit Wait:** Waits for a specific condition to occur before proceeding. Used with WebDriverWait and ExpectedConditions. Applied for specific elements.
     + *When to use:* Implicit wait for general element availability. Explicit wait for specific conditions like visibility, clickability, text presence, etc. Avoid mixing them as it can lead to unpredictable wait times.
5. **How do you handle dynamic web elements in Selenium?**
   * *Expected Answer:* Using dynamic XPath or CSS selectors (e.g., using contains(), starts-with(), or text() in XPath; or attribute selectors in CSS). Using explicit waits.
6. **Explain the Page Object Model (POM) and its benefits.**
   * *Expected Answer:* A design pattern that creates an object repository for web UI elements. Each web page in the application has a corresponding Page Class.
   * *Benefits:* Improved code reusability, readability, maintainability, reduced duplication, easier to understand and update tests.
7. **How do you handle alerts (JavaScript pop-ups) in Selenium?**
   * *Expected Answer:* Using the Alert interface: with below methods.
     + driver.switchTo().alert().accept(): To click OK/Accept.
     + driver.switchTo().alert().dismiss(): To click Cancel/Dismiss.
     + driver.switchTo().alert().getText(): To get the text of the alert.
     + driver.switchTo().alert().sendKeys("text"): To send text to a prompt alert.
8. **How do you handle frames (iframes) in Selenium?**
   * *Expected Answer:* Using driver.switchTo().frame() method:
     + driver.switchTo().frame(int index)
     + driver.switchTo().frame(String nameOrId)
     + driver.switchTo().frame(WebElement frameElement)
     + To switch back to the main content: driver.switchTo().defaultContent()
9. **How do you take screenshots in Selenium?**
   * *Expected Answer:* Using TakesScreenshot interface:

Java

File src = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(src, new File("./Screenshots/error.png")); // Requires Apache Commons IO

**TestNG**

1. **What is TestNG? What are its advantages over JUnit?**
   * *Expected Answer:* A testing framework inspired by JUnit and NUnit, but with more powerful and flexible functionalities.
   * *Advantages:* Rich set of annotations, grouping tests, parallel execution, parameterization, dependency management, powerful reporting.
2. **Explain common TestNG annotations (@Test, @BeforeMethod, @AfterMethod, @BeforeClass, @AfterClass, @BeforeSuite, @AfterSuite, etc.).**
   * *Expected Answer:* Be able to explain the purpose and execution order of these annotations. For example, @Test marks a method as a test method, @BeforeMethod runs before each test method, @BeforeClass runs once before all test methods in a class, etc.
3. **How do you achieve parallel test execution in TestNG?**
   * *Expected Answer:* By configuring the testng.xml file. You can set parallel="methods", parallel="classes", parallel="tests", or parallel="instances" and specify the thread-count.
4. **What are TestNG groups? How do you use them?**
   * *Expected Answer:* A way to categorize and run specific sets of test methods. Defined using the groups attribute in the @Test annotation. Configured in testng.xml to include or exclude groups.
5. **What are TestNG DataProviders? How do you use them for data-driven testing?**
   * *Expected Answer:* A method to supply data to test methods. A method annotated with @DataProvider returns a 2D array of Object[] or Iterator<Object[]>. The @Test method uses the dataProvider attribute to link to it.
6. **Explain assertions in TestNG. What is the difference between Assert.assertEquals() and SoftAssert?**
   * *Expected Answer:*
     + **Assertions:** Validation points in tests to verify expected outcomes.
     + **Assert.assertEquals() (Hard Assert):** If an assertion fails, the test method execution stops immediately.
     + **SoftAssert:** Collects all assertion failures during a test method's execution and reports them at the end of the method. The test continues even if an assertion fails. Requires softAssert.assertAll() at the end.

**Maven**

1. **What is Maven? Why is it used in automation testing?**
   * *Expected Answer:* A build automation tool primarily for Java projects. It manages dependencies, builds projects, and defines project structure.
   * *Use in Automation:* Manages Selenium and TestNG dependencies, compiles test code, runs tests, generates reports, and helps standardize project setup.
2. **What is a POM.xml file? Explain its key elements.**
   * *Expected Answer:* Project Object Model (POM) file. It's an XML file that contains project configuration, dependencies, plugins, build profiles, etc.
   * *Key Elements:* <project>, <modelVersion>, <groupId>, <artifactId>, <version>, <packaging>, <dependencies>, <build>, <plugins>.
3. **How do you add a dependency to your Maven project?**
   * *Expected Answer:* By adding the <dependency> tag with <groupId>, <artifactId>, and <version> inside the <dependencies> section of pom.xml.
4. **Explain the Maven build lifecycle phases (e.g., clean, compile, test, package, install, deploy).**
   * *Expected Answer:*
     + clean: Cleans the project build directory.
     + compile: Compiles the source code.
     + test: Runs the tests.
     + package: Packages the compiled code into a JAR/WAR.
     + install: Installs the package into the local Maven repository.
     + deploy: Deploys the package to a remote repository.

**Git**

1. **What is Git? Why is it used in automation testing?**
   * *Expected Answer:* A distributed version control system (DVCS) used for tracking changes in source code during software development.
   * *Use in Automation:* Collaboration on test scripts, tracking changes, reverting to previous versions, branching for new features/bug fixes, code reviews.
2. **Explain basic Git commands: git clone, git add, git commit, git push, git pull, git branch, git checkout, git merge.**
   * *Expected Answer:*
     + git clone <repository\_url>: Creates a local copy of a remote repository.
     + git add <file>: Stages changes for the next commit.
     + git commit -m "message": Saves staged changes to the local repository.
     + git push: Uploads local commits to a remote repository.
     + git pull: Fetches changes from a remote repository and merges them into the current branch.
     + git branch <branch\_name>: Creates a new branch.
     + git checkout <branch\_name>: Switches to a different branch.
     + git merge <branch\_name>: Integrates changes from one branch into another.
3. **How do you handle merge conflicts in Git?**
   * *Expected Answer:* When Git cannot automatically merge changes (e.g., same lines edited in different branches). Manually resolve conflicts in the affected files, git add the resolved files, and then git commit.

**Scenario-Based Questions**

1. **Describe your approach to automate a login functionality.**
   * *Expected Answer:* (Refer to POM) Identify elements (username, password, login button). Create a LoginPage class with methods to interact with these elements. Write a test case using TestNG to call these methods with test data (possibly from a DataProvider). Include assertions for successful login and error messages.
2. **How do you handle flaky tests in your automation suite?**
   * *Expected Answer:* Identify the root cause (network issues, sync issues, dynamic elements, bad locators). Implement more robust waits (explicit waits), retry mechanisms (TestNG IRetryAnalyzer), improve locators, or add pre-conditions/post-conditions.
3. **How do you decide what to automate and what not to automate?**
   * *Expected Answer:* Automate repetitive tasks, regression tests, data-driven tests, tests that run frequently. Don't automate exploratory testing, usability testing, or tests that require human intuition or are rarely run.
4. **Explain your experience with automation frameworks. What components would you include in a robust framework?**
   * *Expected Answer:* Discuss your experience with Selenium, TestNG, Maven. A robust framework includes:
     + Page Object Model (POM)
     + TestNG for test organization and execution
     + Maven for dependency management and build
     + Reporting (ExtentReports, Allure)
     + Logging (Log4j)
     + Data-driven capabilities (Excel, CSV, DataProviders)
     + Utils/Helper classes (reusable methods)
     + Screenshot on failure

By preparing for these questions, you'll demonstrate a strong understanding of Java, automation principles, and practical application, which is crucial for a 2-year experienced automation tester. Good luck!

Encapulation:

Wrapping of data under a single unit.- secure the data within our class and methods.

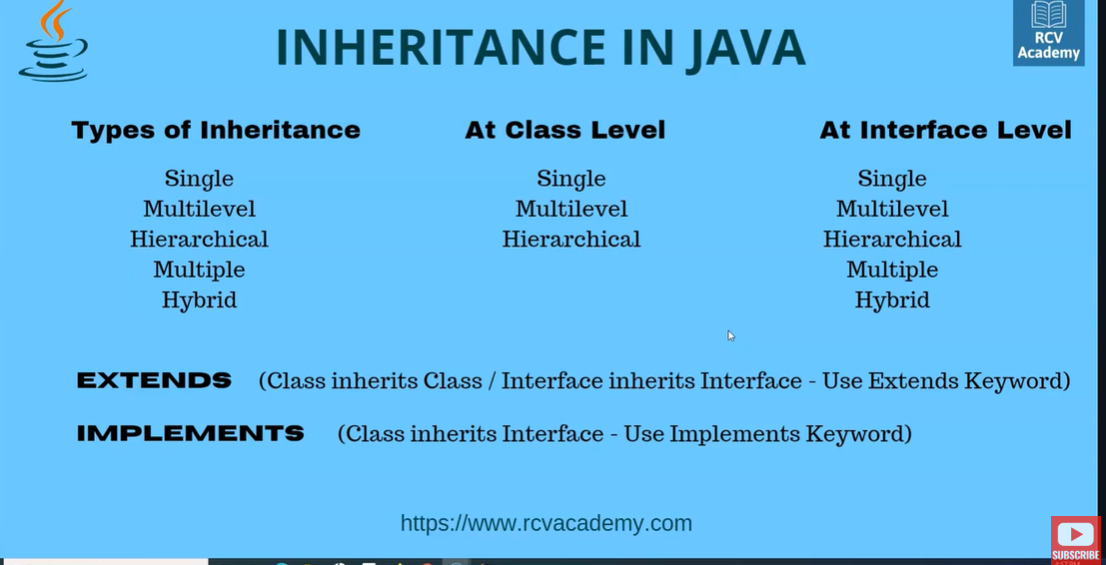
1.- define data as private-

Set and Get data- setter and getter methods.

Ex: trying to develop a game.- when you don’t want to provide invalid values- with encapsulation set and get methods we can have logic to set and Get data.

Advantage – hiding the data- using private. And Reusability- .- imp for selenium.

Java Inheritance and Types:



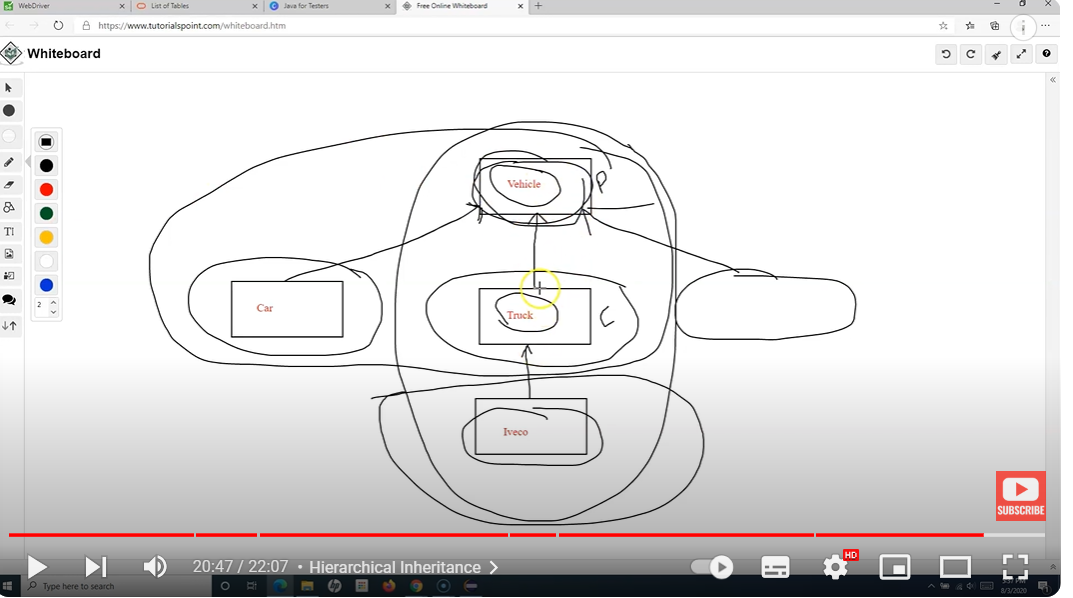
Vehicle- Truck ---- Single Inheritance.

Vehicle- Truck- Inveto- Multilevel Inheritance.

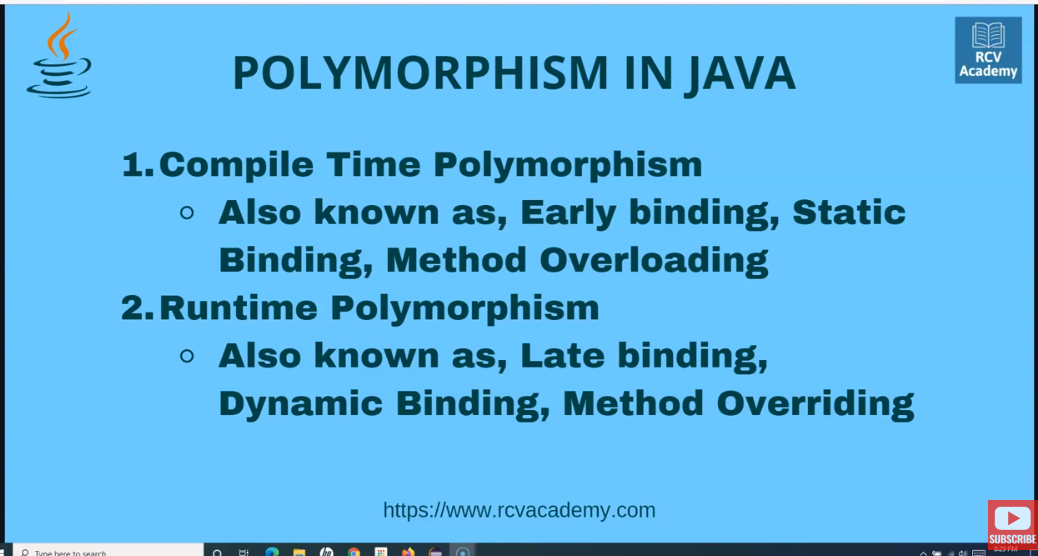
Vehicle--- 1. Truck, 2. Car- Hierarchical Inheritance.

Multiple and Hybrid not used for selenium automation.

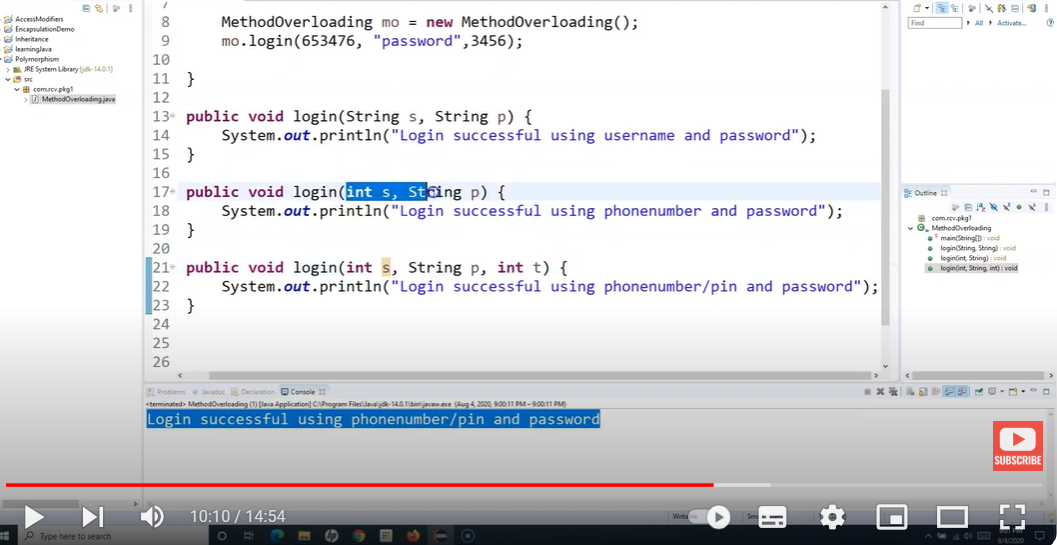
Class B extends Class A.



POLYMORPHISM:



Method Overloading- same method with diff method signatures. Ex: log in to an app with username password, log in with phone and password , username otp etc.

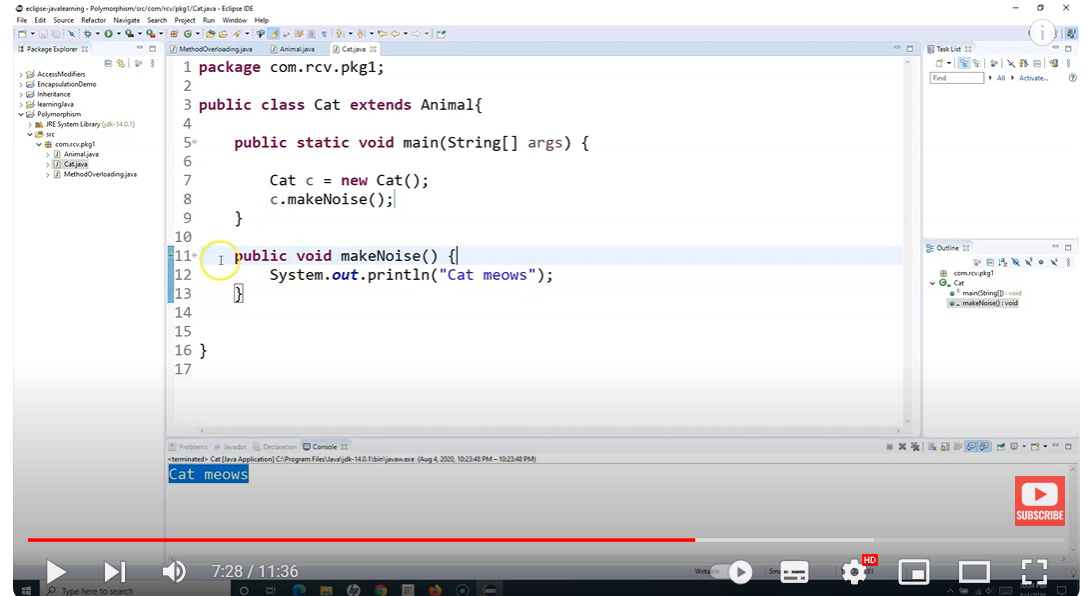


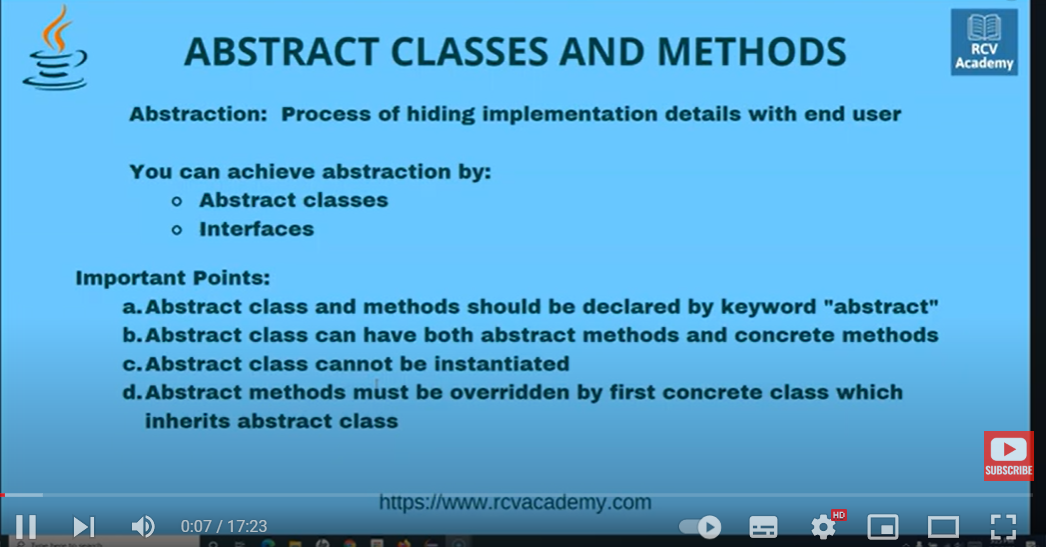
Method Overriding:- Method name needs to be same- parameters/method signature also needs to be same.Method Overriding/Runtime polymorphism is possible only when inheritance is there.

Ex: If there is a parent class- Animal with method make noise in it(prints animal makes noise)

A child class- Cat which extends Animal class and make noise method is called from (Parent class), the o/p printed will be Animal makes noise. But now we are in Cat class and we want something specific there like Cat meows or something. So in that case, we can create the same method again within that child class Cat that prints Cat meows. This way the existing method- makes noise() from parent class is overridden by the makes noise method in child class and the O/p can be overridden.

Imp Rules for overriding: 1.Method must have same name and parameters needs to be same as that of Parent class.2, Inheritance is Must(Is relation is must)- Overriding can be done in child class

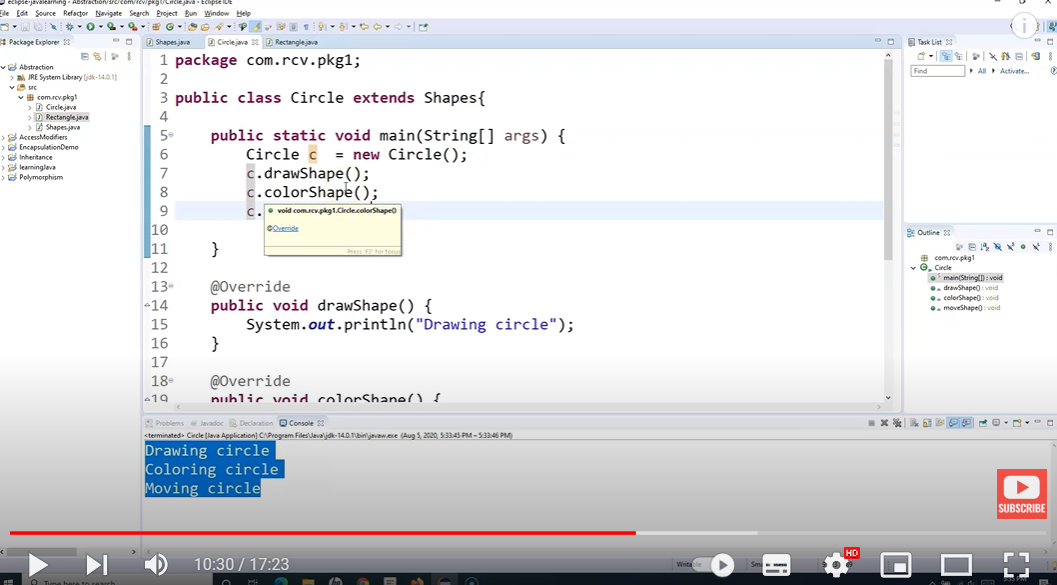




ABSTRACTION: Method of hiding the implementation details with the end user.

Abstract classes.

1. Abstract methods need to be defined with abstract keyword.
2. Abstract methods cannot have method body.
3. The Class within which the abstract method exists that also needs to have keyword abstract. A class can have both abstract and non abstract(concrete) methods- (they are methods which have method body).
4. We define abstract classes when we are only aware of the generic functionality and don’’t exactly know how the o/p is expected. Ex: Shapes, Animals etcs.



Ex Selenium- click method- we don’t know which browser – then define abstract classes.

1. Abstract methods cannot be instantiated.- If there is no inheritance. We create a object of the parent class in child class and call the parent abstract method in child class – its not possible. Throws an error.

INTERFACES in Java: We cannot have any concrete methods in Interface. – that is the diff between abstract class and Interface. WEBDRIVER is Interface in Selenium- all methods are abstracrt.

Constructors in Java: very similar to methods- they do not have any return type and name of the constructor is same as the class name. Constructors are used to create an instance of the class.

