**📌 Java Cheatsheet for Test Automation Engineers**

**1. Java Basics (Syntax & Structure)**

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, Automation!");

}

}

* public class → Every Java program has a class.
* main() → Entry point.
* System.out.println → Prints output.

🔑 **Keywords testers must know:** class, public, static, void, int, String.

**2. Data Types & Variables**

int age = 30;

double salary = 50000.50;

boolean isTester = true;

char grade = 'A';

String name = "Harish";

* **Primitive types:** int, double, boolean, char.
* **Non-primitive type:** String, arrays, objects.

**3. Operators**

int a = 5, b = 2;

System.out.println(a + b); // 7 (Arithmetic)

System.out.println(a > b); // true (Comparison)

System.out.println(a == 5 && b == 2); // true (Logical)

* Arithmetic: + - \* / %
* Comparison: == != > < >= <=
* Logical: && || !

**4. Control Statements**

if (a > b) {

System.out.println("a is greater");

} else {

System.out.println("b is greater");

}

for (int i = 0; i < 5; i++) {

System.out.println(i);

}

while (a > 0) {

a--;

}

* if-else, switch-case
* Loops: for, while, do-while, for-each

👉 Useful in **iterating test data** or handling **conditional checks** in automation.

**5. Arrays & Collections**

// Array

int[] numbers = {1, 2, 3, 4};

// List

List<String> browsers = new ArrayList<>();

browsers.add("Chrome");

browsers.add("Firefox");

// Map

Map<String, String> env = new HashMap<>();

env.put("QA", "qa.example.com");

env.put("Prod", "prod.example.com");

* Array → fixed size.
* List → dynamic, ordered.
* Set → unique values.
* Map → key-value pairs.

👉 Collections are heavily used in **test data storage & retrieval**.

**6. Methods**

public int add(int x, int y) {

return x + y;

}

* Reusable logic.
* Supports **data-driven testing** & **utility functions**.

**7. Object-Oriented Programming (OOP)**

class Car {

String brand;

Car(String brand) { this.brand = brand; }

public void drive() { System.out.println(brand + " is driving"); }

}

Car c = new Car("BMW");

c.drive();

* **Class & Object**
* **Constructor**
* **Encapsulation** → use private + getters/setters
* **Inheritance** → extends
* **Polymorphism** → method overloading/overriding
* **Abstraction** → abstract class or interface

👉 Test frameworks (JUnit, TestNG, Selenium POM) rely heavily on OOP.

**8. Exception Handling**

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

System.out.println("Error: " + e.getMessage());

} finally {

System.out.println("Cleanup!");

}

* try-catch-finally
* Common in **handling automation failures**.

**9. File Handling**

FileReader fr = new FileReader("data.txt");

BufferedReader br = new BufferedReader(fr);

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

👉 Important for **data-driven testing** (CSV, JSON, Excel).

**10. Java + Automation Framework Hooks**

* **Selenium WebDriver**

WebDriver driver = new ChromeDriver();

driver.get("https://example.com");

driver.findElement(By.id("username")).sendKeys("test");

* **TestNG / JUnit**

@Test

public void loginTest() {

Assert.assertEquals(2+2, 4);

}

* **Maven/Gradle** → dependency management.
* **Log4j** → logging test results.

👉 This covers **core Java** that an automation engineer **must know before diving deep into Selenium, TestNG, Cypress (JavaScript), or API testing**.

Perfect 👍  
We already covered **Core Java (Fundamentals)**. Now let’s move to the **Next Segment: Advanced Java for Automation**.

This part is essential because in **real-world test automation**, you’ll handle large test data, functional libraries, reporting, and integrations — which rely on these **advanced Java features**.

**📌 Java Cheatsheet for Test Automation Engineers (Advanced Segment)**

**11. Wrapper Classes & Autoboxing**

int a = 10; // primitive

Integer b = a; // autoboxing (primitive → object)

int c = b; // unboxing (object → primitive)

* Wrapper classes: Integer, Double, Boolean, etc.  
  👉 Useful when working with **Collections**, which don’t accept primitives.

**12. Generics**

List<String> browsers = new ArrayList<>();

browsers.add("Chrome");

browsers.add("Firefox");

for (String b : browsers) {

System.out.println(b);

}

* Generics ensure **type safety**.
* Avoids casting issues.

👉 Often used in **framework libraries & utility functions**.

**13. Lambda Expressions (Java 8+)**

List<String> names = Arrays.asList("Harish", "QA", "Automation");

names.forEach(n -> System.out.println(n));

* -> syntax for **functional programming**.
* Reduces boilerplate code.

👉 Used in Selenium with streams for filtering elements.

**14. Streams API (Java 8+)**

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

// Filter and collect even numbers

List<Integer> even = numbers.stream()

.filter(n -> n % 2 == 0)

.collect(Collectors.toList());

* Powerful for **processing test data**.
* Common in **CSV/Excel parsing & reporting**.

**15. Date & Time (Java 8+)**

LocalDate today = LocalDate.now();

LocalDate future = today.plusDays(10);

System.out.println(today); // 2025-09-03

👉 Needed in automation for **validating timestamps, scheduling, and date pickers**.

**16. Enums**

enum Browser { CHROME, FIREFOX, EDGE }

Browser b = Browser.CHROME;

System.out.println(b);

👉 Used in **cross-browser testing frameworks**.

**17. Static & Final**

class Config {

public static final String BASE\_URL = "https://testsite.com";

}

* static → belongs to class (shared).
* final → constant or unmodifiable.

👉 Used for **constants, framework settings**.

**18. Access Modifiers**

* public → accessible everywhere
* private → within class
* protected → within package + subclass
* (default) → within package

👉 Needed in **Page Object Model (POM)** for structuring automation code.

**19. Multithreading (Basics)**

class TestThread extends Thread {

public void run() {

System.out.println("Running test: " + Thread.currentThread().getName());

}

}

TestThread t1 = new TestThread();

t1.start();

* Used in **parallel execution of tests**.
* TestNG provides parallel="methods" leveraging this.

**20. File I/O (Advanced)**

// Writing to file

FileWriter fw = new FileWriter("output.txt");

fw.write("Automation Results");

fw.close();

// Reading all lines (Java 8+)

List<String> lines = Files.readAllLines(Paths.get("data.txt"));

👉 Needed for **test data input/output & logs**.

**21. Serialization & Deserialization**

// Object to file (Serialization)

ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("obj.dat"));

out.writeObject(myObject);

// File to object (Deserialization)

ObjectInputStream in = new ObjectInputStream(new FileInputStream("obj.dat"));

MyClass obj = (MyClass) in.readObject();

👉 Used in **API testing (JSON, XML mapping)** and storing test states.

**22. Properties & Config Files**

Properties prop = new Properties();

FileInputStream fis = new FileInputStream("config.properties");

prop.load(fis);

System.out.println(prop.getProperty("url"));

👉 Useful for **storing environment configs (QA/Dev/Prod)**.

**23. Excel & CSV Handling**

(Using Apache POI for Excel)

FileInputStream fis = new FileInputStream("data.xlsx");

XSSFWorkbook wb = new XSSFWorkbook(fis);

XSSFSheet sheet = wb.getSheet("TestData");

String value = sheet.getRow(0).getCell(0).getStringCellValue();

👉 Critical for **data-driven automation**.

**24. Logging (Log4j / SLF4J)**

private static final Logger logger = LogManager.getLogger(TestClass.class);

logger.info("Test started");

logger.error("Error occurred");

👉 Helps in **debugging test executions**.

**25. Java + Test Automation Integrations**

* **Reflection API** → dynamic test execution (used in frameworks).
* **Maven/Gradle** → build & dependency management.
* **JUnit/TestNG Annotations** → @BeforeTest, @AfterTest, @DataProvider.
* **POI / OpenCSV / Jackson** → handle data formats (Excel/CSV/JSON).

✅ With this advanced cheatsheet, you’re now set to understand **real-world automation frameworks (Selenium, TestNG, RestAssured, Cucumber)**.

Great 🚀  
We covered **Core Java** and **Advanced Java**. Now let’s wrap it up with the **Final Segment: Java Cheatsheet for Framework Building**.

This will tie everything together into **practical automation use cases** — exactly what you’ll face when building or working with frameworks like Selenium + TestNG + RestAssured + Cucumber.

**📌 Java Cheatsheet for Test Automation Engineers (Framework Segment)**

**26. Project Structure (Maven-style)**

src

├── main

│ └── java → Utilities, Helpers, Core

│ └── resources → config.properties, testdata files

├── test

│ └── java → Test Classes

│ └── resources → testng.xml, log4j2.xml

pom.xml → Dependencies

👉 Always organize code into **layers** (Tests, Pages, Utils).

**27. Page Object Model (POM)**

public class LoginPage {

WebDriver driver;

By username = By.id("user");

By password = By.id("pass");

By loginBtn = By.id("login");

public LoginPage(WebDriver driver) {

this.driver = driver;

}

public void login(String user, String pass) {

driver.findElement(username).sendKeys(user);

driver.findElement(password).sendKeys(pass);

driver.findElement(loginBtn).click();

}

}

👉 Reduces duplication, improves maintainability.

**28. TestNG / JUnit Annotations**

@BeforeClass

public void setup() {

driver = new ChromeDriver();

}

@Test

public void loginTest() {

LoginPage lp = new LoginPage(driver);

lp.login("admin", "password");

Assert.assertEquals(driver.getTitle(), "Dashboard");

}

@AfterClass

public void teardown() {

driver.quit();

}

* @BeforeClass → runs once before all tests.
* @BeforeMethod → runs before each test.
* @DataProvider → supplies test data.
* @Test(priority, groups) → categorizing tests.

**29. Data-Driven Testing (with TestNG DataProvider)**

@DataProvider(name = "loginData")

public Object[][] getData() {

return new Object[][] {

{"admin", "password"},

{"user", "user123"}

};

}

@Test(dataProvider = "loginData")

public void loginTest(String username, String password) {

LoginPage lp = new LoginPage(driver);

lp.login(username, password);

}

👉 Runs the same test with multiple inputs.

**30. Utility Classes (Reusable Helpers)**

public class WaitHelper {

WebDriver driver;

public WaitHelper(WebDriver driver) {

this.driver = driver;

}

public void waitForElement(By locator, int seconds) {

new WebDriverWait(driver, Duration.ofSeconds(seconds))

.until(ExpectedConditions.visibilityOfElementLocated(locator));

}

}

👉 Keep **waits, screenshots, readers, DB connections** in Utils.

**31. Assertions**

Assert.assertTrue(driver.getTitle().contains("Dashboard"));

Assert.assertEquals(actual, expected, "Login failed!");

👉 Core of **validation in automation**.

**32. Parallel Execution (TestNG)**

<suite name="ParallelTests" parallel="tests" thread-count="2">

<test name="ChromeTests">

<parameter name="browser" value="chrome"/>

<classes>

<class name="tests.LoginTest"/>

</classes>

</test>

<test name="FirefoxTests">

<parameter name="browser" value="firefox"/>

<classes>

<class name="tests.LoginTest"/>

</classes>

</test>

</suite>

👉 Achieves **cross-browser, parallel runs**.

**33. Configuration Handling**

config.properties

url=https://testsite.com

browser=chrome

timeout=10

Properties prop = new Properties();

FileInputStream fis = new FileInputStream("config.properties");

prop.load(fis);

String baseUrl = prop.getProperty("url");

👉 Keeps test setup flexible for multiple environments.

**34. Logging & Reporting**

* **Log4j for logging**

logger.info("Test started");

logger.error("Element not found!");

* **ExtentReports / Allure Reports** → generate rich reports for management.

**35. API Testing with RestAssured (Java)**

given()

.baseUri("https://reqres.in")

.when()

.get("/api/users/2")

.then()

.statusCode(200)

.body("data.first\_name", equalTo("Janet"));

👉 For **REST API validation** (common in modern test automation).

**36. Cucumber (BDD in Java)**

* **Feature File**

Feature: Login

Scenario: Valid Login

Given User is on Login Page

When User enters valid credentials

Then User should see Dashboard

* **Step Definition**

@Given("User is on Login Page")

public void user\_on\_login\_page() {

driver.get("https://testsite.com/login");

}

👉 Behavior-driven testing for business alignment.

**37. Continuous Integration (CI/CD)**

* Run tests on **Jenkins/GitHub Actions/GitLab CI**.
* Configure mvn test or gradle test.
* Generate & publish **HTML/XML reports**.

**✅ Wrap-up**

With these **three cheatsheet segments**:

1. **Core Java (fundamentals for testers)**
2. **Advanced Java (real-world automation needs)**
3. **Framework Building (TestNG, Selenium, RestAssured, POM, CI/CD)**

👉 You now have a **practical reference for Java in automation testing**.