# Java Concepts: Detailed Comparison

## Constructors vs. Methods

|  |  |  |
| --- | --- | --- |
| Aspect | Constructor | Method |
| Definition | A block of code that initializes a newly created object. | A collection of statements that returns a value upon execution. |
| Purpose | Used to initialize an object. | Contains Java code to perform operations. |
| Invocation | Invoked implicitly by the system when an object is created using the `new` keyword. | Invoked explicitly through method calls by the programmer. |
| Return Type | Does not have a return type. | Must have a return type. |
| Object State | Initializes an object that doesn't exist. | Performs operations on an already created object. |
| Naming | Must have the same name as the class. | Can have any name. |
| Overloading | A class can have many constructors, but their parameters must differ. | A class can have many methods, but their parameters must differ. |
| Inheritance | Cannot be inherited by subclasses. | Can be inherited by subclasses. |

## Constructor Overloading vs. Method Overloading

|  |  |  |
| --- | --- | --- |
| Aspect | Constructor Overloading | Method Overloading |
| Name | Constructors share the same name as the class. | Methods share the same name within a class. |
| Return Type | Constructors have no return type. | Methods must have a return type. |
| Invocation | Triggered when an object is created using the `new` keyword. | Invoked explicitly using method names. |
| Purpose | Provides multiple ways to initialize objects with different parameters. | Provides multiple ways to perform the same action with different parameters. |
| Parameter List | Requires different argument lists for constructor overloading. | Requires different parameter lists for method overloading. |
| Inheritance | Cannot be inherited, but can be invoked using the `super` keyword. | Can be inherited and overridden in child classes. |
| Default Behavior | The compiler provides a default no-argument constructor if none is defined. | The compiler does not provide a default method if none is declared. |

## Method Overloading vs. Method Overriding

|  |  |  |
| --- | --- | --- |
| Aspect | Method Overloading | Method Overriding |
| Polymorphism | Compile-time polymorphism. | Runtime polymorphism. |
| Purpose | Increases program readability. | Provides a specific implementation of a method defined in the parent class. |
| Class Context | Occurs within a single class. | Requires two classes with an inheritance relationship. |
| Inheritance | Does not require inheritance. | Always requires inheritance. |
| Signatures | Methods must have the same name but different signatures. | Methods must have the same name and the same signature. |
| Return Type | The return type can differ or remain the same. | The return type must be the same or covariant. |
| Binding | Static binding is used. | Dynamic binding is used. |
| Modifiers | Private and final methods can be overloaded. | Private and final methods cannot be overridden. |

## Early Binding vs. Late Binding

|  |  |  |
| --- | --- | --- |
| Aspect | Early Binding | Late Binding |
| Timing | Compile-time process. | Runtime process. |
| Binding | Links method definition and call at compile time. | Links method definition and call at runtime. |
| Object | Does not use the actual object for binding. | Uses the actual object for binding. |
| Examples | Method overloading. | Method overriding. |
| Performance | Faster execution. | Slower execution. |

## Static Polymorphism vs. Dynamic Polymorphism

|  |  |  |
| --- | --- | --- |
| Aspect | Static Polymorphism | Dynamic Polymorphism |
| Definition | Also known as compile-time polymorphism. | Also known as runtime polymorphism. |
| Achieved By | Method overloading. | Method overriding. |
| Binding | Uses compile-time (early) binding. | Uses runtime (late) binding. |
| Performance | Faster than dynamic polymorphism. | Slower than static polymorphism. |
| Inheritance Requirement | Does not require inheritance. | Requires inheritance. |

## Mutable vs. Immutable Objects

|  |  |  |
| --- | --- | --- |
| Aspect | Mutable | Immutable |
| Value Modification | Values can be changed after initialization. | Values cannot be changed after initialization. |
| State Change | The state of the object can change. | The state of the object cannot change. |
| Object Formation | No new object is formed when values are altered. | A new object is formed when values are altered. |
| Methods | Provides methods to change object values. | Does not provide methods to change object values. |
| Getter/Setter | Supports `get()` and `set()` methods for object manipulation. | Only supports `get()` methods. |
| Thread Safety | May or may not be thread-safe. | Always thread-safe. |
| Class Requirements | Requires methods for modifying fields and getters/setters. | Requires a `final` class, `private` fields, and `final` objects. |

**Public Access Modifier vs. Private Access Modifier**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Public Access Modifier** | **Private Access Modifier** |
| **Applicability** | This modifier is applicable for both top-level classes and interfaces. | This modifier is not applicable for both top-level classes and interfaces. |
| **Access from Child Class (Same Package)** | Public members can be accessed from the child class of the same package. | Private members cannot be accessed from the child class of the same package. |
| **Access from Non-Child Class (Same Package)** | Public members can be accessed from non-child class of the same package. | Private members cannot be accessed from non-child class of the same package. |
| **Access from Child Class (Outside Package)** | Public members can be accessed from child class of outside package. | Private members cannot be accessed from child class of outside package. |
| **Access from Non-Child Class (Outside Package)** | Public members can be accessed from non-child class of outside package. | Private members cannot be accessed from non-child class of outside package. |
| **Accessibility** | Public modifier is the most accessible modifier. | Private modifier is the most restricted modifier. |
| **Recommended Usage** | Public modifier is the recommended modifier for methods. | Private modifier is the recommended modifier for data members. |

### Wait() vs. Sleep()

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Wait()** | **Sleep()** |
| **Class** | Wait() method belongs to the **Object** class. | Sleep() method belongs to the **Thread** class. |
| **Lock Behavior** | Wait() releases the lock during synchronization. | Sleep() does not release the lock during synchronization. |
| **Synchronization Context** | Wait() must be called only from a synchronized context. | Sleep() does not require a synchronized context. |
| **Static** | Wait() is not a static method. | Sleep() is a static method. |
| **Overloaded Methods** | **Wait()** has three overloaded methods: | **Sleep()** has two overloaded methods: |
|  | - wait() | - sleep(long millis) (millis: milliseconds) |
|  | - wait(long timeout) | - sleep(long millis, int nanos) (nanos: nanoseconds) |
|  | - wait(long timeout, int nanos) |  |
| **Signature Example** | public final void wait(long timeout) | public static void sleep(long millis) throws InterruptedException |

### ****throw**** vs. ****throws****

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Key Difference** | **throw** | **throws** |
| **1** | **Point of Usage** | The throw keyword is used inside a function. It is used when it is required to throw an exception logically. | The throws keyword is used in the function signature. It is used when the function has some statements that can lead to exceptions. |
| **2** | **Exceptions Thrown** | The throw keyword is used to throw an exception explicitly. It can throw only one exception at a time. | The throws keyword can be used to declare multiple exceptions, separated by a comma. The matched exception is thrown automatically. |
| **3** | **Syntax** | Syntax of throw keyword includes the instance of the Exception to be thrown. It is followed by the instance variable. | Syntax of throws keyword includes the class names of the Exceptions to be thrown. It is followed by exception class names. |
| **4** | **Propagation of Exceptions** | throw keyword cannot propagate checked exceptions. It is only used to propagate unchecked Exceptions that are not checked via throws. | throws keyword is used to propagate checked Exceptions only. |

### Errors vs. Exceptions

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Errors** | **Exceptions** |
| **Recoverability** | Recovering from an error is not possible. | Exceptions can be recovered from using a try-catch block or throwing exceptions back to the caller. |
| **Type** | All errors in Java are of the unchecked type. | Exceptions include both checked and unchecked types. |
| **Cause** | Errors are mostly caused by the environment in which the program is running. | The program itself is responsible for causing exceptions. |
| **Occurrence** | Errors can occur at compile time. | Unchecked exceptions occur at runtime, whereas checked exceptions occur at compile time. |
| **Package** | Errors are defined in the java.lang.Error package. | Exceptions are defined in the java.lang.Exception package. |
| **Examples** | Examples: java.lang.StackOverflowError, java.lang.OutOfMemoryError. | **Checked Exceptions**: SQLException, IOException. **Unchecked Exceptions**: ArrayIndexOutOfBoundsException, NullPointerException, ArithmeticException |