Main Method & Command Line Arguments

The `main()` method is the **entry point of execution** for a Java application.
Understanding its **exact signature, JVM behavior, and edge cases** is **high-weight** in Oracle Java exams.

---

1. Responsibility of Compiler vs JVM

- The **compiler does NOT check**:
  - Whether a class has a `main()` method
  - Whether the `main()` method is correctly declared
- The **JVM checks this at runtime**

If the JVM **cannot find a valid `main()` method**, the program **compiles successfully** but fails at runtime.

Runtime Error

```
Error: Main method not found in class Test
```

✔ This is a **runtime error**, not a compile-time error
✔ Very commonly tested in MCQs

---

2. Exact Signature Required by JVM

The JVM **strictly searches** for the following method:

```
public static void main(String[] args)
```

Why each keyword matters

Keyword	Reason
`public`	JVM must access it from anywhere

`static`	JVM should call it without creating an object
`void`	JVM does not expect a return value
`String[] args`	Command line arguments

❌ Any mismatch → **runtime error**, not compile error

---

3. Acceptable Variations (Exam-Safe)

The following variations are **100% valid**:

Modifier Order

```
static public void main(String[] args)
```

String Array Declaration Styles

```
String[] args
String []args
String args[]
```

Parameter Name

```
public static void main(String[] values)
```

✔ Parameter name **does not matter**

---

4. `main()` with Var-Args (Java 5+)

Java allows **var-args** in `main()`:

```
public static void main(String... args)
```

✔ JVM treats `String... args` as `String[] args`

---

5. Allowed Modifiers for `main()`

The `main()` method **can** use:

- `final`
- `synchronized`
- `strictfp`

```
public static final synchronized strictfp void main(String... args)
```

✔ Still a valid entry point

---

6. Common Invalid `main()` Declarations (Very Important)

Declaration	Reason
`public void main(String[] args)`	Not static
`public static int main(String[] args)`	Return type not void
`public static void Main(String[] args)`	Case-sensitive
`public static void main(String args)`	Not an array
`public static void main(Object[] args)`	JVM won’t call it

⚠️ **All of these compile successfully**, but fail at runtime.

---

7. Overloading the `main()` Method

Is it allowed?

✔ **Yes**, `main()` can be overloaded.

Will JVM call overloaded versions?

❌ **No**. JVM **always calls only**:

```
main(String[] args)
```

Example

```
public static void main(int[] args) {
    System.out.println("int[] main");
}
```

✔ This method is valid
❌ JVM will never call it automatically

---

8. Inheritance & `main()` Method

Key Rule

`main()` is **static**, so it is **not overridden** — it is **hidden**.

Case 1: Child does NOT define `main()`

```
class Parent {
    public static void main(String[] args) {
        System.out.println("Parent main");
    }
}

class Child extends Parent {}
```

```
java Child
```

✔ Output:

```
Parent main
```

Case 2: Child defines its own `main()`

```
class Child extends Parent {
    public static void main(String[] args) {
        System.out.println("Child main");
    }
}

java Child
```

✔ Output:

```
Child main
```

> ✔ This is **method hiding**, not overriding
> ✔ Static methods do not participate in runtime polymorphism

---

9. Java 1.7+ Change (Highly Relevant)

Before Java 1.7

- Static blocks executed **even without `main()`**
- Then JVM threw `NoSuchMethodError`

From Java 1.7 onwards

- JVM **requires `main()` first**
- Static blocks will **NOT execute** if `main()` is missing

```
class Test {
    static {
        System.out.println("static block");
    }
}
```

❌ Output (Java 7+):

```
Error: main method not found
```

✔ This change is **very important for exams**

---

10. Order of Execution (Modern Java)

If both exist:

1. Static blocks
2. `main()` method

```
class Test {
    static {
        System.out.println("static block");
    }
    public static void main(String[] args) {
        System.out.println("main method");
    }
}
```

✔ Output:

```
static block
main method
```

---

11. Command Line Arguments

Definition

Arguments passed from the command prompt are called **command line arguments**.

```
java Test A B C
```

Inside Java:

```
args[0] → "A"
args[1] → "B"
args[2] → "C"
args.length → 3
```

---

12. Important Rules About `args`

- `args` is **never null**
- If no arguments are passed → `args.length == 0`
- All arguments are **Strings**

---

13. Common Runtime Exception (Classic Trap)

```
for (int i = 0; i <= args.length; i++) {
    System.out.println(args[i]);
}
```

❌ Causes:

```
ArrayIndexOutOfBoundsException
```

✔ Correct:

```
i < args.length
```

---

14. Reassigning `args` Inside `main()`

```
String[] data = {"X", "Y", "Z"};
args = data;
```

✔ Perfectly valid
✔ `args` is just a local reference variable

15. String Concatenation vs Arithmetic

```
System.out.println(args[0] + args[1]);
```

```
java Test 10 20
```

✔ Output:

```
1020
```

✔ All command line arguments are **Strings**
✔ `+` performs **String concatenation**

16. Arguments with Spaces

Use **double quotes**:

```
java Test "Sai Charan"
```

Inside Java:

```
args[0] → "Sai Charan"
```

17. Which `main()` Does JVM Call? (Trick Question)

```
public static void main(int[] args) {}
public static void main(Object[] args) {}
public static void main(String[] args) {}
```

```
java Test 1 2 3
```

✔ JVM always calls:

```
main(String[] args)
```

18. Certification Summary (Must-Remember)

- JVM checks `main()` **at runtime**
- Signature must be **exact**
- `main()` can be overloaded but **not overridden**
- Static methods use **method hiding**
- Java 7+ requires `main()` even for static blocks
- Command line arguments are **Strings**
- `args.length` is critical to avoid runtime errors

Additional Modern Relevance

- In **real projects**, `main()` often delegates to frameworks:
    - Spring Boot → `SpringApplication.run()`
    - JavaFX → `Application.launch()`
- Oracle exams still test **raw JVM behavior**, not frameworks
- Understanding `main()` is essential for:
    - Debugging startup issues
    - JVM internals
    - Interview questions