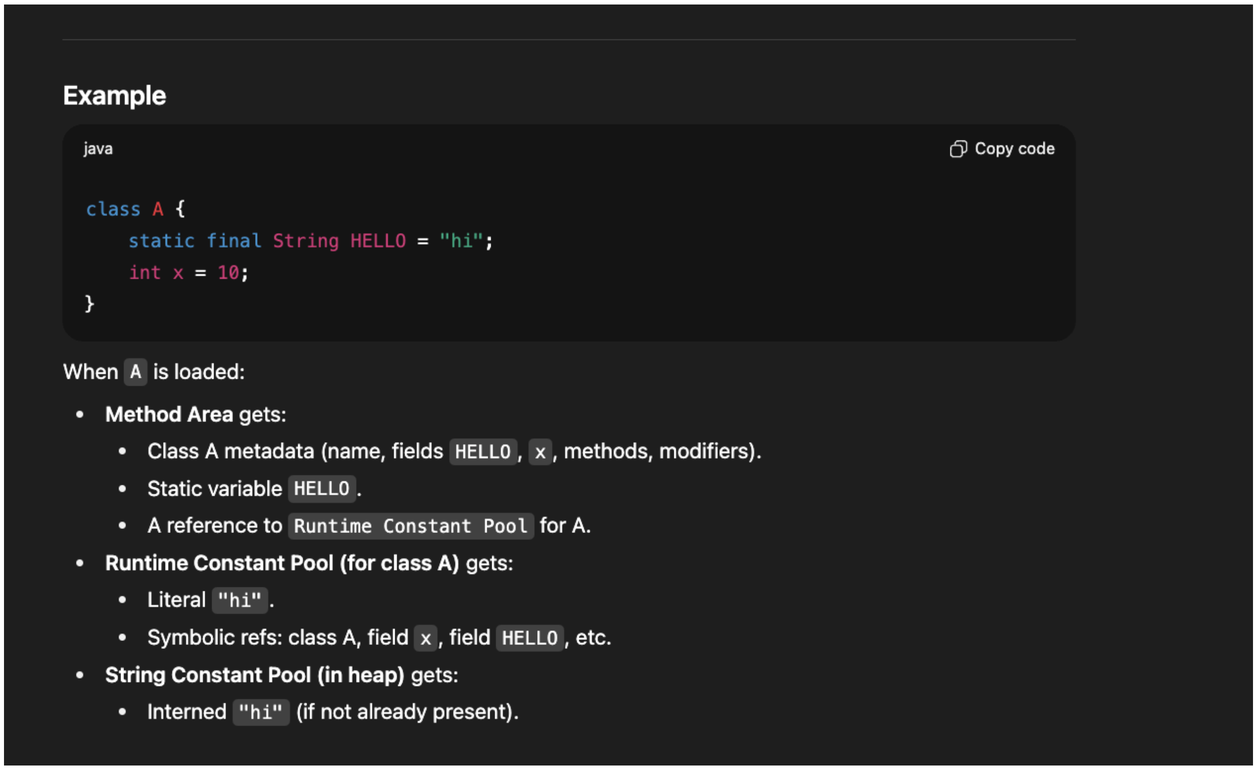
1. Method Area (store class metadata), Runtime Constant Pool (bind value), String Constant Pool



Example

java Copy code

```java
class A {
    static final String HELLO = "hi";
    int x = 10;
}
```

When `A` is loaded:

- **Method Area** gets:
  - Class A metadata (name, fields `HELLO`, `x`, methods, modifiers).
  - Static variable `HELLO`.
  - A reference to `Runtime Constant Pool` for A.
- **Runtime Constant Pool (for class A)** gets:
  - Literal `"hi"`.
  - Symbolic refs: class A, field `x`, field `HELLO`, etc.
- **String Constant Pool (in heap)** gets:
  - Interned `"hi"` (if not already present).

2. Program Counter, Java Virtual Machine Stack, Native Method Stack, Heap

- Program Counter: per thread, store the last executed command.
- Java Virtual Machine Stack: per thread, stack to store variables.
- Native Method Stack: stack of C/C++ programs.
- Heap: store object

3. JVM Steps During Object Creation (with `new`)

1. **Class loading & linking (if not already loaded)**
    - ClassLoader loads the `.class` file.
    - JVM verifies, prepares, and resolves the class.
2. **Memory allocation (Heap)**
    - JVM allocates memory for the new object.
    - Memory size = sum of instance fields + object header.
3. **Default initialization**
    - All fields initialized to default values (`0`, `null`, `false`).
4. **Constructor execution**
    - Explicit constructor runs.
    - Instance variables initialized.
5. **Reference assignment**
    - The variable (`p` in our case) stores the reference (on the stack).
    -

4. Memory Layout of an Object

Each object in JVM typically has:

1. **Object Header**
    - Mark Word (hash code, GC info, lock info).
    - Class pointer (points to class metadata in Method Area/Metaspace).
2. **Instance Data**
    - Values of instance fields.
3. **Padding**
    - To align object size to 8 bytes.