Replace Magic Number with Symbolic Constant

Replace **unnamed numeric literals** with **named constants** for clarity.

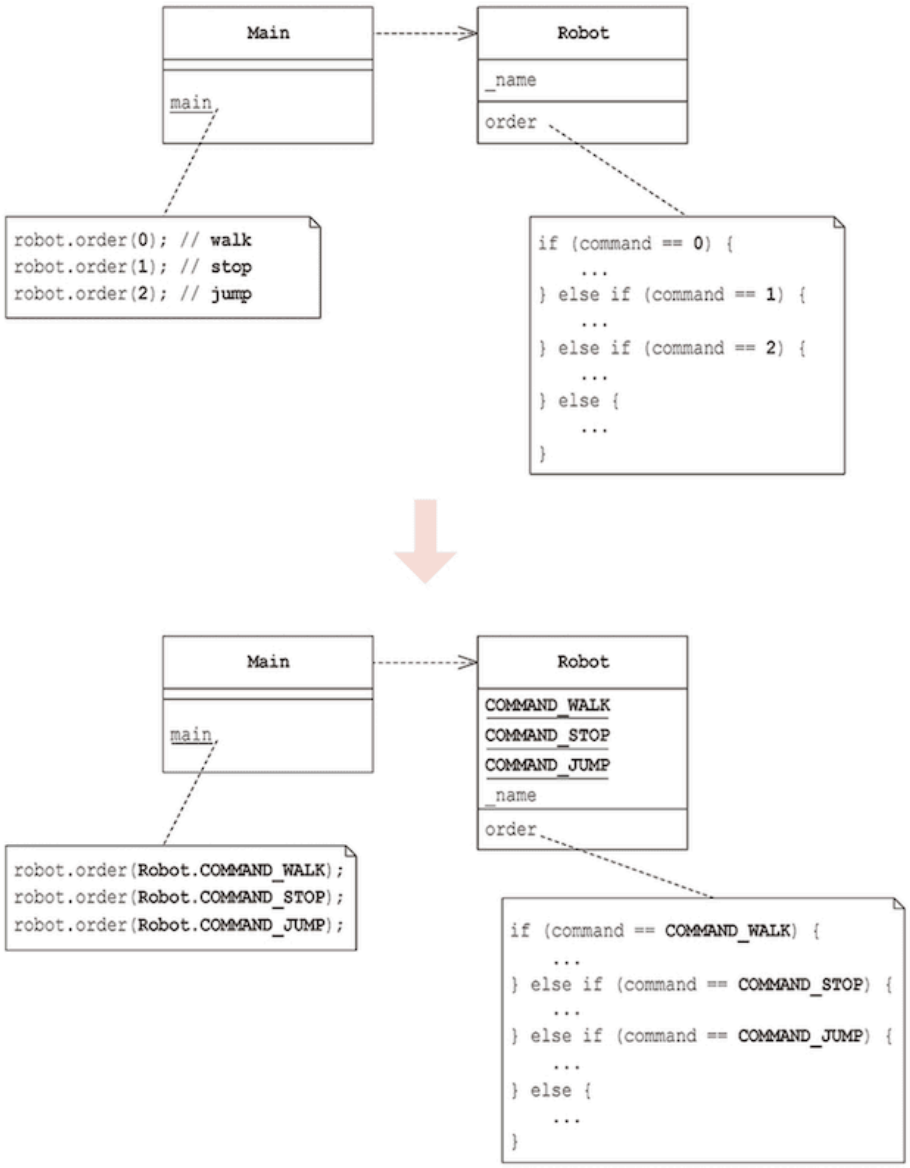
Code Smell

```
if 100 < input.length(): ... # 100???

=>

if MAX_INPUT_LENGTH < input.length(): ... # aha!
```

Example

- Code smell: magic number!

```
def main():
    robot = Robot("Andrew")
    robot.order(0)  # walk - magic number!
    robot.order(1)  # stop - magic number!
    robot.order(2)  # jump - magic number!
```


Main
main
Robot
_name
order
robot.order(0); // walk
robot.order(1); // stop
robot.order(2); // jump
if (command == 0) {
...
} else if (command == 1) {
...
} else if (command == 2) {
...
} else {
...
}
Main
main
Robot
COMMAND_WALK
COMMAND_STOP
COMMAND_JUMP
_name
order
robot.order(Robot.COMMAND_WALK);
robot.order(Robot.COMMAND_STOP);
robot.order(Robot.COMMAND_JUMP);
if (command == COMMAND_WALK) {
...
} else if (command == COMMAND_STOP) {
...
} else if (command == COMMAND_JUMP) {
...
} else {
...
}

```
class Robot:
    COMMAND_WALK = 0
    COMMAND_STOP = 1
    COMMAND_JUMP = 2
...
def main():
    robot = Robot("Andrew")
    robot.order(Robot.COMMAND_WALK)
    robot.order(Robot.COMMAND_STOP)
    robot.order(Robot.COMMAND_JUMP)
```

Unit Test

```
class MainTest(StandardOutputTest):
    def test_main(self):
        # Expected
        expected = self.get_expected_output(
            "Andrew walks.",
            "Andrew stops.",
            "Andrew jumps.")

        # Actual
        main()
        actual = self.get_actual_output()

        # Assert
        self.assertEqual(expected, actual)
```

Tip

Redirect the std output

- In the example, we use the `main()` for generating output.
    - We need to capture the output from it, and compare the output with expected value.
- We use `StandardOutputTest` to capture outputs.

Capturing and testing standard output

```
def setUp(self):
    """Set up output capture"""
    # Save original output stream
    self.saved_stdout = sys.stdout
    # Create a string buffer
    self.actual_output = io.StringIO()
    # Redirect output to our buffer
    sys.stdout = self.actual_output
def tearDown(self):
    """Restore original stdout"""
    sys.stdout = self.saved_stdout
```

Get actual output and expected output.

```
def get_actual_output(self) -> str:
    """Get the captured output"""
    # Make sure all output is written
    sys.stdout.flush()
    # Get the captured string
    return self.actual_output.getvalue()

def get_expected_output(self, *lines: str) -> str:
    """Build expected output from lines"""
    # Join lines with newlines
    return '\n'.join(lines) + '\n'
```

Use the code in "01 common/StandardOutputTest".

Enum

- We can isolate the command even further using Enum.

```
class Robot:
    class Command(Enum):
        WALK = "walk"
        STOP = "stop"
        JUMP = "jump"

def main():
    robot = Robot("Andrew")
    robot.order(Robot.Command.WALK)
    robot.order(Robot.Command.STOP)
    robot.order(Robot.Command.JUMP)
```

Discussion

No change in Test

- We do not change anything in Unit Test.
- We change tests when:
  - we find a bug, but the bug was not found with the test.
  - we add new features, and we add code for the features.

When refactoring magic numbers to symbolic constants, you should change the unit tests.

False: Unit tests should not change because the external behavior remains the same