Replace Type Code with Class

Replace primitive **type codes** with a **class** for clarity and safety.

Type Code

- A **type code** is when we use numbers (or strings) to represent categories of things.

Example:

```
Book = 1
DVD = 2
Software = 3
```

Here, 1, 2, and 3 are type codes.

Why It’s a Problem?

Easy to confuse meaning:

```
type = 233234  # Valid number, but nonsense
type += 1      # Accidentally changes type
```

- Hard to read and debug:
- What does 2 mean? Book or DVD?
- Code becomes less clear.

Example: Item

Before: Item class using raw integer type codes

```python
class Item:
    # Type code constants
    TYPECODE_BOOK = 0
    TYPECODE_DVD = 1
    TYPECODE_SOFTWARE = 2

    def __init__(self, typecode: int, title: str, price: int):
        self.typecode = typecode
        self.title = title
        self.price = price

    def get_typecode(self) -> int:
        return self.typecode
    ...
```

We can use the Item class by specifying the type with the type code.

```python
from Item import Item

def main():
    book = Item(
        Item.TYPECODE_BOOK, "World History", 4800)

    dvd = Item(
        Item.TYPECODE_DVD, "New York Dreams Special Edition", 3000)
    ...
    print(f"book = {book}")
    print(f"dvd  = {dvd}")

if __name__ == "__main__":
    main()
```

After: We introduce a class to replace/hide the type code.

```python
class ItemType:
    def __init__(self, typecode: int):
        self._typecode = typecode
    def get_typecode(self) -> int:
        return self._typecode
# Type-safe constants
ItemType.BOOK = ItemType(0)
ItemType.DVD = ItemType(1)
ItemType.SOFTWARE = ItemType(2)
```

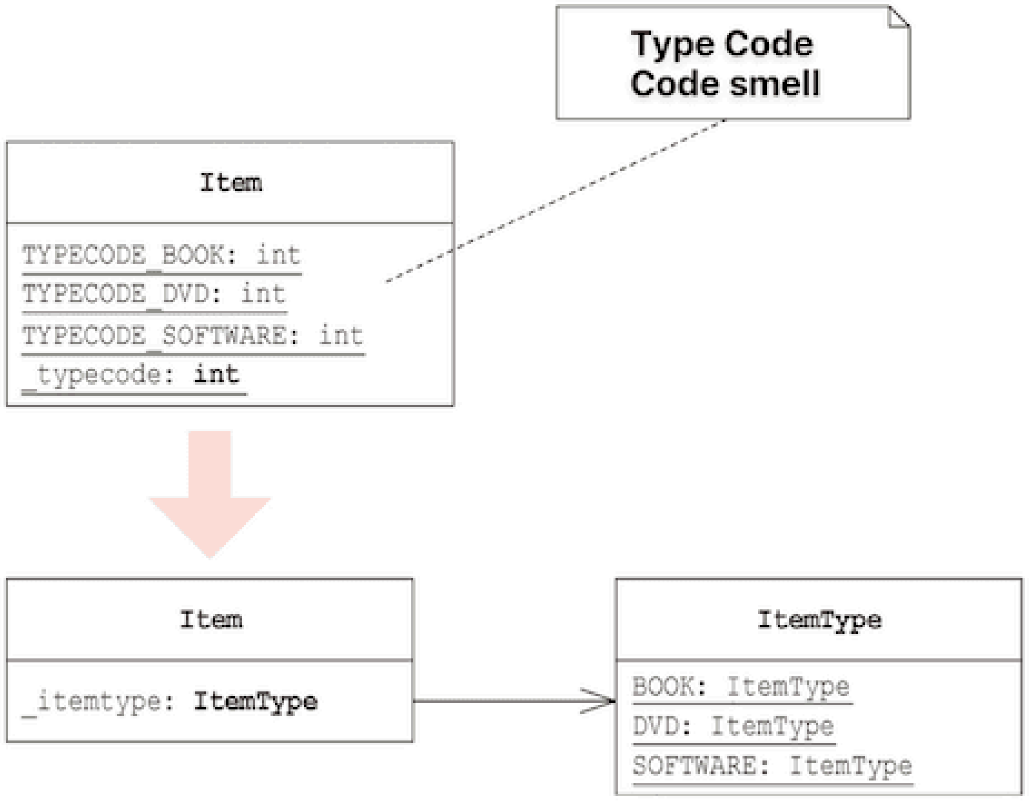
The test code:

```
 def test_create_book(self):
     book = Item(ItemType.BOOK, # <--
       "Python Programming", 1500)

     self.assertEqual(book.get_typecode(), 0)
=>
def test_create_book(self):
     book = Item(Item.TYPECODE_BOOK,
       "Python Programming", 1500)

     self.assertEqual(book.get_typecode(), 0)
```

The type code is encapsulated in the class.

```python
def test_type_constants(self):
    self.assertEqual(Item.TYPECODE_BOOK, 0)
    self.assertEqual(Item.TYPECODE_DVD, 1)
    self.assertEqual(Item.TYPECODE_SOFTWARE, 2)

def test_type_constants(self):
    self.assertEqual(ItemType.BOOK.get_typecode(), 0)
    self.assertEqual(ItemType.DVD.get_typecode(), 1)
    self.assertEqual(ItemType.SOFTWARE.get_typecode(), 2)
```

UML


Type Code
Code smell
Item
TYPECODE_BOOK: int
TYPECODE_DVD: int
TYPECODE_SOFTWARE: int
_typecode: int
Item
_itemtype: ItemType
ItemType
BOOK: ItemType
DVD: ItemType
SOFTWARE: ItemType

Discussion

The benefits of **replacing Type Code with a Class**.

1. **Type safety** - compiler/interpreter can catch type errors
2. **Self-documenting** - code intent is clearer
3. **Extensibility** - easier to add new types
4. **Validation** - can enforce valid values
5. **Encapsulation** - type-related behavior can be added to the class

You should use **Replace Type Code with Class**

- Type code **doesn't affect behavior** (no conditional logic based on type)
- You want **type safety** and better code documentation
- You need to **prevent invalid values**
- Type code is used in **multiple places**