Mixing Static and Non-Static Features

Ahmed Sabbir Arif

Invocation Counter

- We incorporate an invocation counter when we want to keep count of the number of times a method in a class in invoked.
- Why?
  - How popular the method is?
  - The cost of the method?

Example

- Lets assume, we want to keep track of the number of times `getArea` method is used.
- Attribute section:

```
private static int count = 0;
```

- To the body of `getArea`:

```
Rectangle.count++;
```

Example, cont.

- An accessor:

```
public static int getCount()
{
  return Rectangle.count;
}
```

Serial Number on Objects

- Two parts:
  1. Similar to the invocation count,
  2. Stamping the generated serial number on the instance.
- Two attributes:

```
private static int count = 0;
private int number;
```

Serial Number on Objects, cont.

```
public Rectangle(int width, int height)
{
  this.setWidth(width);
  this.setHeight(height);
  Rectangle.count++;
  this.setNumber(Rectangle.count);
}
```

Maintaining a Singleton

- A method:

```java
public static ClassName getInstance()
```

```java
private static ClassName instance = new ClassName(...);
```

```java
public static ClassName getInstance()
{
  return ClassName.instance;
}
```

Maintaining a Singleton, cont.

```
private static Rectangle instance = new Rectangle();

public static Rectangle getInstance()
{
  return Rectangle.instance;
}
```

Maintaining a Singleton, cont.

```
private static Rectangle instance = null;

public static Rectangle getInstance()
{
  if (Rectangle.instance == null)
  {
   Rectangle.instance = new Rectangle();
  }
 return Rectangle.instance;
 }
```

One Instance Per State

- To ensure, we have one instance of a class for every possible combination attribute values.
  - Singleton for every state.

An Example

```
String s1 = "York";
String s2 = "York";
output.println(s1.equals(s2) + " - " + (s1 == s2));
```

- The output is: `true – true`
- The `String` objects `s1` and `s2` are not different objects residing at different addresses in memory!
- Compiler does this (for string literals only) to save memory.

How to Enforce?

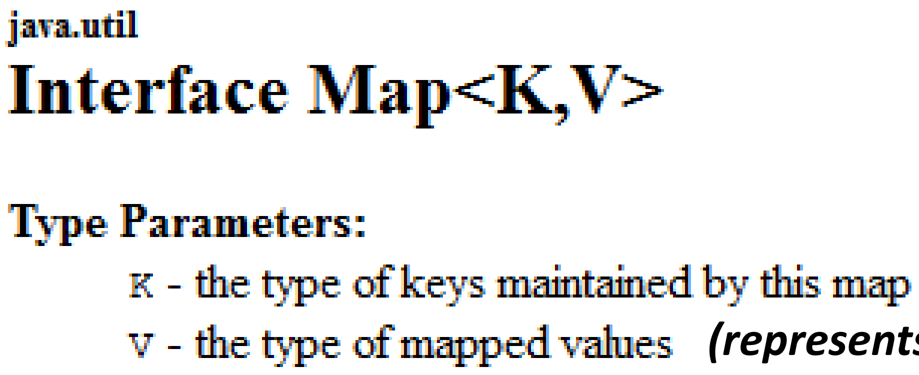
- *Private* constructors (like singleton):
  - Prevents clients from controlling instantiation.
- *Private* mutators (becomes immutable):
  - Prevents clients from changing the state (later).

`getInstance` Method

- For singleton we provided a *static* method, i.e. `getInstance`.
- Here, we will allow clients to pass arguments to it to specify the desired state for the requested instance.
  - *return* if the desired state has been created.
  - otherwise, *create* using a private constructor, *store*, and then *return*.

Storing the Instances

- The *attribute* to store the instances has to be
  - Static, and
  - Able to hold many instances (many states).
  - Therefore, a collection, i.e. Map.


java.util

Interface Map<K,V>

Type Parameters:

K - the type of keys maintained by this map *(represents the state)*

V - the type of mapped values *(represents the instance that has that state)*

Rectangle Example

- We want a `Rectangle` class that allows only one instance for a given *width* and *height*.

interface

```
private static Map<String, Rectangle> instances =
      new TreeMap<String, Rectangle>();
```

Implementing class

  - The map is empty.
  - Now, make the constructors and the mutators.

Rectangle Example, cont.

- A function that stores in the map: a unique key that identifies the state.

```
public static Rectangle getInstance(int width, int height)
{
    String key = width + "-" + height;
    Rectangle instance = Rectangle.instances.get(key);
    if (instance == null)
    {
      instance = new Rectangle(width, height);
      Rectangle.instances.put(key, instance);
    }
    return instance;
}
```

Avoiding Code Duplication

- Three different techniques:
  1. Constructor chaining
  2. Delegation to mutators
  3. Delegation to accessors

Immutable Objects

- It is an object whose state cannot be modified after it is created.
  - String, Double, Integer
- No public mutators

Hash Code Examples

Object	Hash Code
`Point p1 = new Point();`	1
`Point p2 = new Point();`	12

✗

Object	Hash Code
`Point p1 = new Point();`	1
`Point p2 = new Point();`	1

✓

Hash Code Examples

Object	Hash Code
`Point p1 = new Point(1,2);`	1
`Point p2 = new Point(1,1);`	12

✓

Object	Hash Code
`Point p1 = new Point(1,2);`	1
`Point p2 = new Point(1,1);`	1

✓