CS 635 Advanced Object-Oriented Design & Programming

Spring Semester, 2006
Doc 15 Prototype
Apr 6, 2006

How do test code based on random numbers?

```
import java.util.Random;

public class RandomExample {
    Random source = new Random();
    public boolean isFoo(int x) {
        int cutOff = source.nextInt(10);
        return x < cutOff;
    }
}
```

```
import java.util.Random;

public class RandomExample {
	Random source = new Random();

	Random newRandom() {
		return new Random();
	}

	public boolean isFoo(int x) {
		int cutOff = source.nextInt(10);
		return x < cutOff;
	}
}

class TestableRandom extends RandomExample {
	Random newRandom() {
		return new Random((long)0.5);
	}
}
```

It this testable?

Prototype

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype

Applicability

Use the Prototype pattern when

A system should be independent of how its products are created, composed, and represented; and

When the classes to instantiate are specified at run-time; or

To avoid building a class hierarchy of factories that parallels the class hierarchy of products; or

When instances of a class can have one of only a few different combinations of state.

Insurance Example

Insurance agents start with a standard policy and customize it

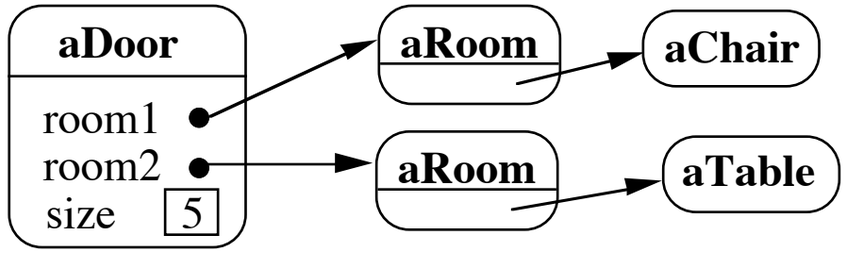
Two basic strategies:

- Copy the original and edit the copy
- Store only the differences between original and the customize version in a decorator

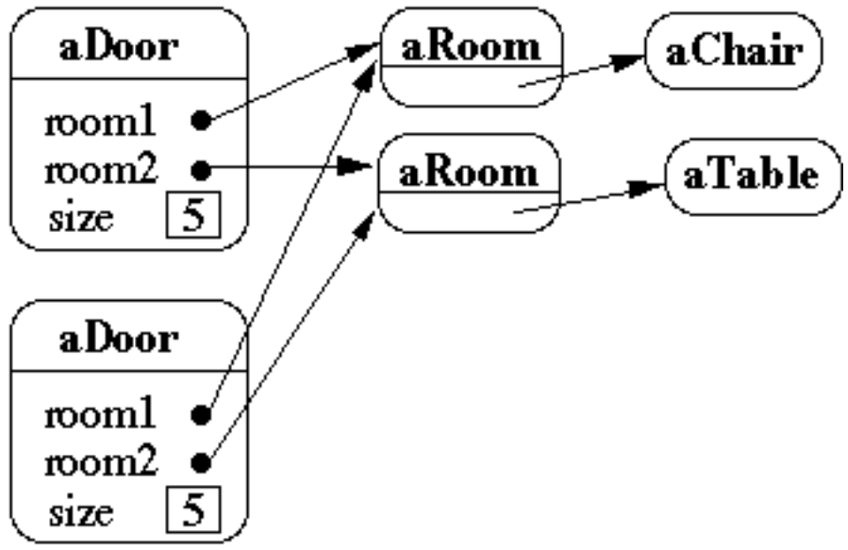
Copying Issues

Shallow Copy Verse Deep Copy

Original Objects

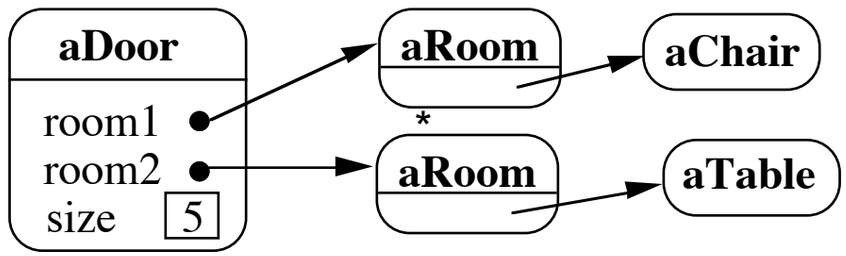


Shallow Copy

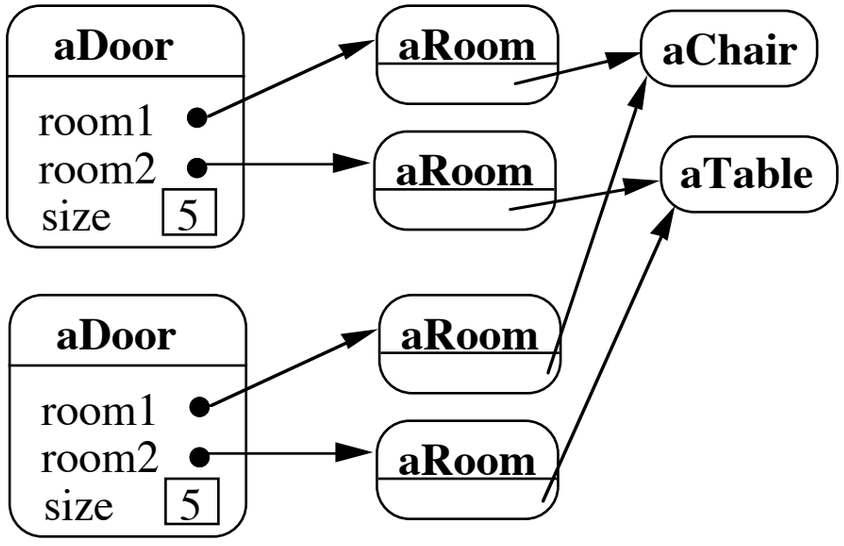


Shallow Copy Verse Deep Copy

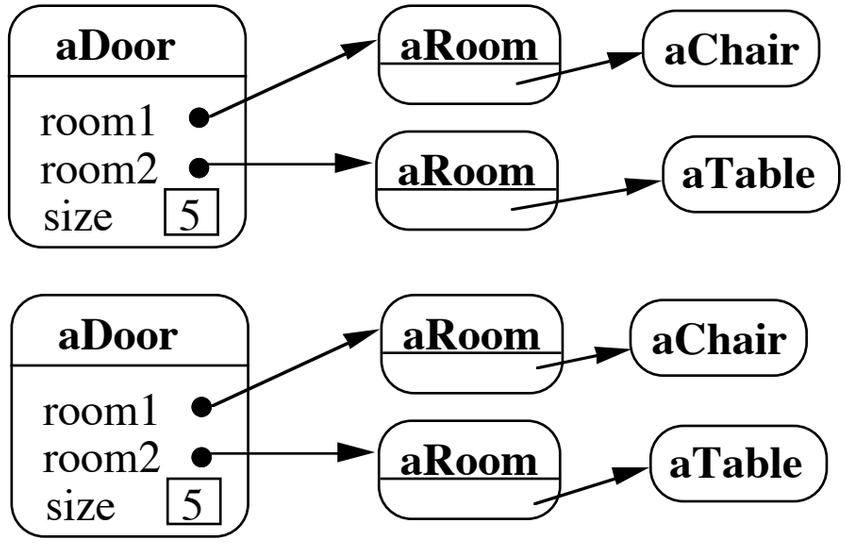
Original Objects



Deep Copy



Deeper Copy



Cloning Issues - C++ Copy Constructors

```
class Door {
        public:
                Door();
                Door( const Door&);
                virtual Door* clone() const;

                virtual void Initialize( Room*, Room* );
                // stuff not shown
        private:
                Room* room1;
                Room* room2;
        }

Door::Door ( const Door& other ) //Copy constructor {
        room1 = other.room1;
        room2 = other.room2;
        }

Door* Door::clone()  const {
        return new Door( *this );
        }
```

Cloning Issues - Java Clone

Shallow Copy

```
class Door implements Cloneable {
        private Room room1;
        private Room room2;

        public Object clone() throws  CloneNotSupportedException {
                return super.clone();
        }
}
```

Deep Copy

```
public class Door implements Cloneable {
        private Room room1;
        private Room room2;

        public Object clone() throws CloneNotSupportedException {
                Door thisCloned =(Door) super.clone();
                thisCloned.room1 = (Room)room1.clone();
                thisCloned.room2 = (Room)room2.clone();
                return thisCloned;
        }
}
```