

# Creating Classes Part I

1

---

---

---

---

---

---

---

---

## Agenda

- Defining simple classes.
- Instance variables and methods.
- Objects.
- Object references.

2

---

---

---

---

---

---

---

---

## Reading

- You should be reading:
  - Chapter 6 of ‘Developing Java Software’

3

---

---

---

---

---

---

---

---

## Classes & Objects

- A class is a template or blueprint for a particular type of object.
- An object is an instance of a class.
- We design programs in terms of classes and objects.
- We write the classes using the programming language.
- The running program creates, destroys and manipulates objects in memory.

4

---

---

---

---

---

---

---

---

## Classes

- The contents and behaviour of a type of an object are described in its class.
- A class consists of:
  - A collection of *instance variables* that contain the object's data.
  - A collection of *methods* that implement the operations.

5

---

---

---

---

---

---

---

---

## A very simple example

2 instance variables,  
2 methods

```
/** ... */
public class Book
{
    private int shelfCode = 0 ;
    private String title = "Unnamed" ;

    /** ... */
    public int getShelfCode() { return code ; }
    public String getTitle() { return title ; }
}
```

Note - no static

6

---

---

---

---

---

---

---

---

## Javadoc Comments

- All classes start with a javadoc comment that says what they do:

```
/**
 * <dl>
 * <dt>Purpose:
 * <dd> Simple example book class.
 *
 * <dt>Description:
 * Contains a String (title) and an int (code) with methods to get their values.
 * <dd>
 * </dd>
 * </dl>
 *
 * @author Danny Alexander
 * @version $Date: 2002/6/28$
 */
```

7

---

---

---

---

---

---

---

---

## Javadoc comments

- Every method in a class should have its own javadoc comment:

```
/**
 * Returns the shelf code.
 */
public int getShelfCode() {
    return code ;
}
/**
 * Returns the title.
 */
public String getTitle() {
    return name ;
}
```

8

---

---

---

---

---

---

---

---

## Class MyClass

- By convention a class name always starts with a capital letter.
- *Objects* are *instances* of the class and we create them using:  
    Book aBook = new Book() ;
- You can create as many objects as you want.

9

---

---

---

---

---

---

---

---

## A Book Object

- Contains two data.
- Can be used with the methods in its class.

```
aBook : Book
shelfCode = 0
title =
  "Unnamed"
```

10

---

---

---

---

---

---

---

---

## Objects

- Each object has its own private set of variables.

```
a : Book
code = 172
name = "The Confusion"
```

```
c : Book
code = 241
name = "noddy"
```

```
b : Book
code = 9
name = "Moby Dick"
```

11

---

---

---

---

---

---

---

---

## Private

- A class defines a scope.
- Declaring a method or variable private means that it can only be accessed within the scope of the class.
  - This means within a method body or an initialisation expression (constructor) belonging to the class.

12

---

---

---

---

---

---

---

---

## Public

---

- Methods and variables declared public can be accessed by anything that has a reference to an object of the class.
- They form the *public interface* of objects of the class.

13

---

---

---

---

---

---

---

---

## Encapsulation

---

- Public and private are how encapsulation is enforced.
- Good practice states:
  - Instance variables are *always* private.
  - Only a minimal number of methods should be made public.

14

---

---

---

---

---

---

---

---

## Calling methods

---

- We can call the public methods of this object:

```
Book aBook = new Book();  
int n = aBook.getShelfCode();  
String s = aBook.getTitle();
```
- Only methods declared in class Book can be called.

15

---

---

---

---

---

---

---

---

## Method parameters

- Methods can have parameters and return a result.
- We have seen methods that return a result: `getShelfCode()` and `getTitle()`.
- Lets add two methods to make our class more interesting...

16

---

---

---

---

---

---

---

---

## Method Parameters

```
/**
 * Sets the shelf code.
 */
public void setShelfCode(int n) {
    shelfCode = n ;
}

/**
 * Sets the title.
 */
public void setTitle(String s) {
    title = s;
}
```

17

---

---

---

---

---

---

---

---

## Method Parameters

- Now we can change the values contained in particular objects:
- ```
Book b1 = new Book();
Book b2 = new Book() ;
b1.setShelfCode(1);
b1.setTitle("Jack & Jill");
b2.setShelfCode(2);
b2.setTitle("Ulysses");
```

18

---

---

---

---

---

---

---

---

## Type MyClass

- Book is a *new type*.
- This is why declarations such as:  

```
Book aBook = new Book();
```

are possible.
- In fact, Book is a *User Defined Type*.

---

---

---

---

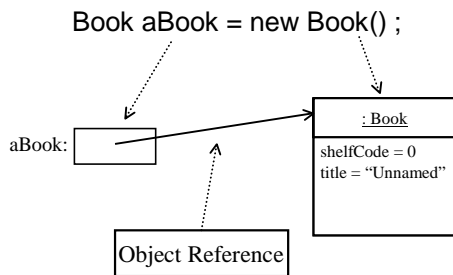
---

---

---

---

## Object references (difficult but important!)



---

---

---

---

---

---

---

---

## References

- A variable of a non-primitive type holds a *reference* to an object.
- It does not hold the object itself.
- The variable can go out of scope but the object can still exist (provided it is referenced by some other variable).
- One object can be referenced by many references and, hence, variables.

---

---

---

---

---

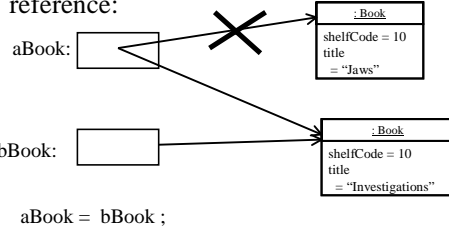
---

---

---

## Assignment

- Assignment changes the value of the reference:



22

---

---

---

---

---

---

---

---

## To summarise

- When you declare a variable of non-primitive type, you get a container that holds an object reference.
- Assigning an “object to a variable” means storing a reference to the object in the variable.

23

---

---

---

---

---

---

---

---

## Object Parameters

- Suppose you want to pass a Book object as a parameter to a method:  

```
void someMethod(Book bbb)
{
    ...
}
```
- A parameter variable can be declared as normal.

24

---

---

---

---

---

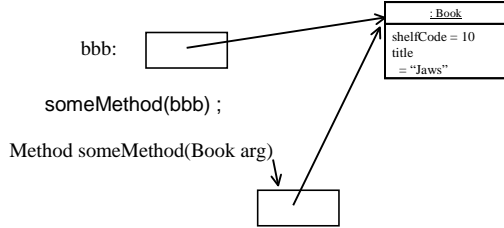
---

---

---

## Parameters & References

- But, consider what happens:



25

---

---

---

---

---

---

---

---

## Object Parameters

- The parameter value is an *object reference*, not an object.
- The parameter variable is initialised to hold a *copy* of the reference.
- The object is *not* copied.

26

---

---

---

---

---

---

---

---

## Consequences

- If an object reference is passed as a parameter then:
  - Changing the object inside the method changes the object outside the method.
  - They are the same object!
  - But changing which object is referenced within the method doesn't change anything outside the method.

27

---

---

---

---

---

---

---

---

## Remember Primitive Types?

- Values of primitive types (int, char, long, boolean, etc.) are stored directly in variables using a binary representation.
- They are *not* objects.
- You cannot have references to primitive types.

28

---

---

---

---

---

---

---

---

## Primitive type parameters

- When a value of a primitive type is passed as a parameter, it is *always* copied.
- The parameter variable is initialised to a copy of the argument value.

29

---

---

---

---

---

---

---

---

## Primitives and Objects

- Anything with *non*-primitive type in Java:
  - Arrays
  - Java objects (e.g., String, Random)
  - Objects of user defined type
- Is accessed via an object reference.
- The rules above apply for passing these types of object between functions.

30

---

---

---

---

---

---

---

---

## Call-by-value

- The parameter passing mechanism used by Java is called “Call-by-value”.
- The value of a parameter is always copied and a parameter variable initialised with the copy.
- Objects are not passed as parameters, only references.
- Objects are “passed by reference”.

31

---

---

---

---

---

---

---

---

## Garbage Collection

- What happens to the object referenced by s?

```
public void message() {  
    String s = "Hello";  
    System.out.println(s);  
}
```

- It is reclaimed by the *Garbage Collector*.
- Once an object is not referenced it is reclaimed.

32

---

---

---

---

---

---

---

---

## Summary

- Seen how to construct very simple classes.
- Methods and instance variables.
- Object references.
- References and parameter passing.

33

---

---

---

---

---

---

---

---