CONSTRUCTORS

zyBook 9.4, zyBook 9.5, zyBook 9.8
Oracle - Java Tutorial: Providing Constructors for Your Classes

@ Dr. Jessica Young Schmidt and NCSU Computer Science Faculty

CONSTRUCTORS

Constructor → A special method that initialized the state of new objects as they are created:

```
public <class name>(<type> <name>, ..., <type> <name>) {
        <statement>;
        ...
        <statement>;
}
```

- Name matches class name
- No return type
- Parameters that specify object's initial state
- Access object's fields and methods directly
- Once we write a constructor, we can no longer use default constructor (constructor automatically supplied by Java with no parameters). If we need a constructor with no parameters in addition to the other constructors we create, we have to define it ourselves.

CONSTRUCTING A BOOK OBJECT

Constructing a Book Object

```
Book book = new Book ();      // Book() with no parameters – default constructor
book.setAuthor("Andrea Beaty");
book.setTitle("Rosie Revere , Engineer");
book.setPubYear(2013);
```

Constructor:

```
Book book1 = new Book(title, author, pubYear, loc);
Book book2 = new Book(title, author, pubYear);
```

State:

Type	Field Name	Description
`String`	`title`	Name of the book
`String`	`author`	Author of the book
`int`	`pubYear`	Publication year of the book
`String`	`checkedOut`	Person who has checked out the book
`int`	`location`	Location of the book in the stacks
`Date`	`dueDate`	Date a checked out book is due

Behavior:

Return Type	Method Name	Description
`boolean`	`checkOut(unityId)`	Check out a book
`void`	`checkIn()`	Check in a book
`int`	`getLocation()`	Info on where the book is stored
`void`	`setLocation(loc)`	Put book at this location in the library

```
public Book(String title, String author, int pubYear, int location) {
   this.title = title;
   this.author = author;
   this.pubYear = pubYear;
   this.location = location;
   checkedOut = null;
   dueDate = null;
}
```

Object Construction

Book b = new Book("Rosie Revere, Engineer", "Andrea Beaty", 2013, 7);

- Creates a Book reference, b
- Creates a Book object by calling Book constructor with given parameters
- Assigns the newly create object to be stored in reference variable b

COMMON CONSTRUCTOR BUGS

Re-declaring fields as local variables ("shadowing")

- This declares local variables with the same name as the fields, rather than storing values into the fields. The fields remain 0 or null.

```
public class Book {
    private String title;
    private String author;
    private int pubYear;
    private String checkedOut;
    private int location;
    private Date dueDate;

    public Book(String newTitle, String newAuthor,
                int newPubYear, int newLocation) {
        String title = newTitle;
        String author = newAuthor;
        int pubYear = newPubYear;
        int location = newLocation;
        String checkedOut = null;
        Date dueDate = null;
    }
}
```

Accidentally giving the constructor a return type

- This is actually not a constructor, but a method named Book.

```
public class Book {
    private String title;
    private String author;
    private int pubYear;
    private String checkedOut;
    private int location;
    private Date dueDate;

    public void Book(String newTitle, String newAuthor,
                     int newPubYear, int newLocation) {
        title = newTitle;
        author = newAuthor;
        pubYear = newPubYear;
        location = newLocation;
        checkedOut = null;
        dueDate = null;
    }
}
```

MULTIPLE CONSTRUCTORS

A class can have multiple constructors and each one must accept a unique set of parameters.

```
// Constructor with parameters
public Book(String title, String author, int pubYear, int location) {
  this.title = title;
  this.author = author;
  this.pubYear = pubYear;
  this.location = location;
  checkedOut = null;
  dueDate = null;
}

// Constructor with parameters but no location
public Book(String title, String author, int pubYear) {
  this(title, author, pubYear, 0);
}
```

Programming Paradigm

- One constructor contains true initialization code – all other constructors call it

Constructor with No Parameters

- Only create constructor with no parameters if it makes sense for your object.
  - We expect each Book to have at a minimum a title, author, and publication year. Therefore, we do not include a constructor with no parameters.

CLASS INVARIANTS

- A property that is true of every object of a class
- An assertion about an object's state that is true for the lifetime of that object
- Encapsulated objects can have constraints on their state
- Can enforce within public methods that client uses to modify object
- Can enforce encapsulation with private methods that are not available to client

BOOK CLASS INVARIANTS

- All locations will be non-negative
  - enforced in setter method that is called from constructor
- Title and author will be non-null and contain at least one non-whitespace character
  - enforced in setter method that is called from constructor
- Title and author cannot be modified after Book is constructed
  - enforced with private setter method, which is not accessible to client

```
import java.util.Date;

public class Book {
    private int id;
    private String title;
    private String author;
    private int pubYear;
    private String checkedOut;
    private int location;
    private Date dueDate;

    public Book(String title, String author, int pubYear, int location) {
        this.setTitle(title);
        this.setAuthor(author);
        this.pubYear = pubYear;
        this.setLocation(location);
        checkedOut = null;
        dueDate = null;
    }

    public Book(String title, String author, int pubYear) {
        this(title, author, pubYear, 0);
    }

    public void setLocation(int location) {
        if (location < 0) {
            throw new IllegalArgumentException("Invalid location");
        }
        this.location = location;
    }
```

```
    private void setTitle(String title) {
        if (title == null) {
            throw new IllegalArgumentException("Invalid title - null");
        }
        // removes leading trailing whitespace
        title = title.trim();
        if (title.length() == 0) {
            throw new IllegalArgumentException("Invalid title - all whitespace");
        }
        this.title = title;
    }

    private void setAuthor(String author) {
        if (author == null) {
            throw new IllegalArgumentException("Invalid author - null");
        }
        // removes leading trailing whitespace
        author = author.trim();
        if (author.length() == 0) {
            throw new IllegalArgumentException("Invalid author - all whitespace");
        }
        this.author = author;
    }

    // Other methods

}
```