CS 241 — Introduction to Problem Solving and Programming

Object-Oriented Programming

First look at classes

Feb 28, 2005

Types

A type is a category of data. It determines

- how much memory an element of this type takes up
- how to interpret data so that it is used as an element of this type

Types

Primitive types: int, double, boolean, char, long, short, float, byte

Array types are different:

- Composite types
- Reference types

Arrays

Problem:

Write a program for a library circulation program. Model books and patrons, keeping track essential information about books (name, author, call number, pages, . . .) and patrons (name, id, . . .), as well as information about how the books and patrons interact (for books, the patron currently holding it and the due date; for patrons, the books currently checked out and any fines).

Library example

If we were to make a composite type for book, it would need

- A title
- An author
- The number of pages
- A call number
- A patron who currently has the book checked out
- The number of days until it is due back.

Library example

If we were to make a composite type for book, it would need

- A title `String`
- An author
- The number of pages
- A call number
- A patron who currently has the book checked out
- The number of days until it is due back.

Library example

If we were to make a composite type for book, it would need

- A title `String`
- An author `String`
- The number of pages
- A call number
- A patron who currently has the book checked out
- The number of days until it is due back.

Library example

If we were to make a composite type for book, it would need

- A title `String`
- An author `String`
- The number of pages `int`
- A call number
- A patron who currently has the book checked out
- The number of days until it is due back.

Library example

If we were to make a composite type for book, it would need

- A title `String`
- An author `String`
- The number of pages `int`
- A call number `int`
- A patron who currently has the book checked out
- The number of days until it is due back.

Library example

If we were to make a composite type for book, it would need

- A title `String`
- An author `String`
- The number of pages `int`
- A call number `int`
- A patron who currently has the book checked out `???`
- The number of days until it is due back.

Library example

If we were to make a composite type for book, it would need

- A title `String`
- An author `String`
- The number of pages `int`
- A call number `int`
- A patron who currently has the book checked out `???`
- The number of days until it is due back. `int`

Library example

If we were to make a composite type for patron, it would need

- A name
- An id
- The amount owed in fines
- The books currently checked out

Library example

If we were to make a composite type for patron, it would need

- A name `String`
- An id
- The amount owed in fines
- The books currently checked out

Library example

If we were to make a composite type for patron, it would need

- A name `String`
- An id `int`
- The amount owed in fines
- The books currently checked out

Library example

If we were to make a composite type for patron, it would need

- A name `String`
- An id `int`
- The amount owed in fines `int`
- The books currently checked out

Library example

If we were to make a composite type for patron, it would need

- A name `String`
- An id `int`
- The amount owed in fines `int`
- The books currently checked out `Book[]`

Library example

If we were to make a composite type for book, it would need

- A title `String`
- An author `String`
- The number of pages `int`
- A call number `int`
- A patron who currently has the book checked out `Patron[]`
- The number of days until it is due back. `int`

Arrays

Arrays have two major draw backs:

- All elements must be the same type
- Elements cannot be associated by a name, only a number.

Records

In general, constructs like this in programming languages are called struct(ure)s or records, and their components are called fields.

The construct in Java that gives this capability (an a lot more) is the class. Its components are called instance variables, and they look like any other variables.

```
public class Class_Name {
        Type Instance_Variable;
        ...
}
```

Book

```
public class Book {
    String title;

    String author;

    int pages;

    int callNumber;

    Patron user;

    int daysTillDue;
}
```

Patron

```
public class Patron {
    String name;

    int id;

    int fines;

    Book[] checkedOut;
}
```

Classes

Like arrays, classes are reference types. A variable with a class type is a reference to an area in memory.

And object is an area of computer memory which can be referred to by a reference an interpreted as an element of a type. An array is an example of an object.

Create a new object of a class using `new`. Such an object is an instance of a class; creating a new on is instantiation.

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

Why the parentheses? It's actually a method, as we'll see later. . .

Classes

Use the instance variables by means of dot notation on an expression which produces a reference to an object (that is, an expression with a class type).

```
Book book = new Book();
book.title = "The Aeneid";
book.author = "Virgil";
Patron patron = new Patron();
patron.name = "Ovid";
patron.checkedOut = new Book[5];
patron.checkOut[0] = book;
System.out.println(patron.name + " has checked out "
                   + patron.checkedOut[0].name);
```

Arrays

Now understand why `length` does not have parentheses for arrays.

```
int[] array;
...

int n = array.length;
```

`length` is an instance variable, specially defined for any array.

Classes

Like arrays. . .

- Using == will compare locations in memory, not contents.
- An assignment to another variable will result in aliasing.
- Assigning `null` means the variable refers to nothing (and attempting to read an instance variable will produce a `NullPointerException`.

Practical details

The class is the main unit of modularity in Java.

Put each class in a separate file, called *Class_Name*`.java`.

Some classes have main methods.

`Library.java:`

```
public class Library {
   public static void main(String[] args) {
...
```

`Book.java:`

```
public class Book {
...
```

Practical Details

Naming convention: Class names should be like variable names except the first letter should be capitalized.

Documentation: Each instance variable should be documented like other variables, but use block style,

```
public class Patron {
    /**
     * The patron's name
     */
    String name;

    /**
     * The patron's id (also serves as index into the
     * array of patrons in the driver)
     */
    int id;
```

Library example

```
public class Library {
    public static void main(String[] args) {
        Book[] shelf = new Book[50];
        Patron[] membership = new Patron[20];

        for(;;) {
            System.out.println("1. Add a new book to the shelf");
            System.out.println("2. Add a new patron");
            System.out.println("3. Check out a book");
            System.out.println("4. Return a book");
            System.out.println("5. Pay fines");
            System.out.println("6. Start a new day");
            System.out.println("7. Quit");
```

Library example

```
            int query = DocsIO.readint("Your choice--> ");
            switch(query) {
            case 1: addBook(shelf); break;
            case 2: addPatron(membership); break;
            case 3: checkout(shelf, membership); break;
            case 4: returnBook(membership); break;
            case 5: payFine(membership); break;
            case 6: newDay(shelf); break;
            case 7: return;
            default: System.out.println("Invalid choice.");
            }
        }
    }
```

Library example

```
static void addBook(Book[] shelf) {
    Book book = new Book();
    book.callNumber = 0;
    while (shelf[book.callNumber] != null) {
        book.callNumber++;
        if (book.callNumber > shelf.length) {
            System.out.println("Shelf full.");
            return;
        }
    }
    shelf[book.callNumber] = book;
    book.title = DocsIO.readString("Title? ");
    book.author = DocsIO.readString("Author? ");
    book.pages = DocsIO.readint("Number of pages? ");
}
```

Library example

```
static void addBook(Book[] shelf) {
    Book book = new Book();
    book.callNumber = 0;
    while (shelf[book.callNumber] != null) {
        book.callNumber++;
        if (book.callNumber > shelf.length) {
            System.out.println("Shelf full.");
            return;
        }
    }
    shelf[book.callNumber] = book;
    book.title = DocsIO.readString("Title? ");
    book.author = DocsIO.readString("Author? ");
    book.pages = DocsIO.readint("Number of pages? ");
}
```

Library example

```
static void addBook(Book[] shelf) {
    Book book = new Book();
    book.callNumber = 0;
    while (shelf[book.callNumber] != null) {
        book.callNumber++;
        if (book.callNumber > shelf.length) {
            System.out.println("Shelf full.");
            return;
        }
    }
    shelf[book.callNumber] = book;
    book.title = DocsIO.readString("Title? ");
    book.author = DocsIO.readString("Author? ");
    book.pages = DocsIO.readint("Number of pages? ");
}
```

Library example

```
static void addBook(Book[] shelf) {
    Book book = new Book();
    book.callNumber = 0;
    while (shelf[book.callNumber] != null) {
        book.callNumber++;
        if (book.callNumber > shelf.length) {
            System.out.println("Shelf full.");
            return;
        }
    }
    shelf[book.callNumber] = book;
    book.title = DocsIO.readString("Title? ");
    book.author = DocsIO.readString("Author? ");
    book.pages = DocsIO.readint("Number of pages? ");
}
```

Library example

```
static void addBook(Book[] shelf) {
    Book book = new Book();
    book.callNumber = 0;
    while (shelf[book.callNumber] != null) {
        book.callNumber++;
        if (book.callNumber > shelf.length) {
            System.out.println("Shelf full.");
            return;
        }
    }
    shelf[book.callNumber] = book;
    book.title = DocsIO.readString("Title? ");
    book.author = DocsIO.readString("Author? ");
    book.pages = DocsIO.readint("Number of pages? ");
}
```

Library example

```
static void addBook(Book[] shelf) {
    Book book = new Book();
    book.callNumber = 0;
    while (shelf[book.callNumber] != null) {
        book.callNumber++;
        if (book.callNumber > shelf.length) {
            System.out.println("Shelf full.");
            return;
        }
    }
    shelf[book.callNumber] = book;
    book.title = DocsIO.readString("Title? ");
    book.author = DocsIO.readString("Author? ");
    book.pages = DocsIO.readint("Number of pages? ");
}
```

Library example

```
static void addPatron(Patron[] membership) {
    Patron patron = new Patron();
    patron.id = 0;
    while (membership[patron.id] != null) {
        patron.id++;
        if (patron.id > membership.length) {
            System.out.println("Membership rolls full.");
            return;
        }
    }
    membership[patron.id] = patron;
    patron.name = DocsIO.readString("Name?" );
    int numBooks =
        DocsIO.readint("How many books may this patron check out at once? ");
    patron.checkedOut = new Book[numBooks];
}
```

Library example

```
static void addPatron(Patron[] membership) {
    Patron patron = new Patron();
    patron.id = 0;
    while (membership[patron.id] != null) {
        patron.id++;
        if (patron.id > membership.length) {
            System.out.println("Membership rolls full.");
            return;
        }
    }
    membership[patron.id] = patron;
    patron.name = DocsIO.readString("Name?" );
    int numBooks =
        DocsIO.readint("How many books may this patron check out at once? ");
    patron.checkedOut = new Book[numBooks];
}
```

Library example

```
static void addPatron(Patron[] membership) {
    Patron patron = new Patron();
    patron.id = 0;
    while (membership[patron.id] != null) {
        patron.id++;
        if (patron.id > membership.length) {
            System.out.println("Membership rolls full.");
            return;
        }
    }
    membership[patron.id] = patron;
    patron.name = DocsIO.readString("Name?" );
    int numBooks =
        DocsIO.readint("How many books may this patron check out at once? ");
    patron.checkedOut = new Book[numBooks];
}
```

Library example

```
static void addPatron(Patron[] membership) {
    Patron patron = new Patron();
    patron.id = 0;
    while (membership[patron.id] != null) {
        patron.id++;
        if (patron.id > membership.length) {
            System.out.println("Membership rolls full.");
            return;
        }
    }
    membership[patron.id] = patron;
    patron.name = DocsIO.readString("Name?" );
    int numBooks =
        DocsIO.readint("How many books may this patron check out at once? ");
    patron.checkedOut = new Book[numBooks];
}
```

Library example

```
static void checkout(Book[] shelf, Patron[] membership) {
    int patronId = DocsIO.readint("Patron id? ");
    if (patronId < 0 || patronId > membership.length ||
        membership[patronId] == null) {
        System.out.println("Patron nonexistant");
        return;
    }
    int checkoutNumber = 0;
    while (membership[patronId].checkedOut[checkoutNumber] != null) {
        checkoutNumber++;
        if (checkoutNumber > membership[patronId].checkedOut.length) {
            System.out.println(membership[patronId].name +
                               " cannot check out any more books.");
            return;
        }
    }
```

Library example

```
static void checkout(Book[] shelf, Patron[] membership) {
    int patronId = DocsIO.readint("Patron id? ");
    if (patronId < 0 || patronId > membership.length ||
        membership[patronId] == null) {
        System.out.println("Patron nonexistant");
        return;
    }
    int checkoutNumber = 0;
    while (membership[patronId].checkedOut[checkoutNumber] != null) {
        checkoutNumber++;
        if (checkoutNumber > membership[patronId].checkedOut.length) {
            System.out.println(membership[patronId].name +
                               " cannot check out any more books.");
            return;
        }
    }
```

Library example

```
        int callNum = DocsIO.readint("Book call number? ");
        if (callNum < 0 || callNum > shelf.length || shelf[callNum] == null) {
            System.out.println("Book nonexistant");
            return;
        }
        if (shelf[callNum].user != null) {
            System.out.println("Book already checked out.");
            return;
        }
        shelf[callNum].daysTillDue =
            DocsIO.readint("How many days until due? ");
        shelf[callNum].user = membership[patronId];
        membership[patronId].checkedOut[checkoutNumber] =
            shelf[callNum];
    }
```

Library example

```
        int callNum = DocsIO.readint("Book call number? ");
        if (callNum < 0 || callNum > shelf.length || shelf[callNum] == null) {
            System.out.println("Book nonexistant");
            return;
        }
        if (shelf[callNum].user != null) {
            System.out.println("Book already checked out.");
            return;
        }
        shelf[callNum].daysTillDue =
            DocsIO.readint("How many days until due? ");
        shelf[callNum].user = membership[patronId];
        membership[patronId].checkedOut[checkoutNumber] =
            shelf[callNum];
    }
```

Library example

```
    int callNum = DocsIO.readint("Book call number? ");
    if (callNum < 0 || callNum > shelf.length || shelf[callNum] == null) {
        System.out.println("Book nonexistant");
        return;
    }
    if (shelf[callNum].user != null) {
        System.out.println("Book already checked out.");
        return;
    }
    shelf[callNum].daysTillDue =
        DocsIO.readint("How many days until due? ");
    shelf[callNum].user = membership[patronId];
    membership[patronId].checkedOut[checkoutNumber] =
        shelf[callNum];
}
```

Library example

```
        int callNum = DocsIO.readint("Book call number? ");
        if (callNum < 0 || callNum > shelf.length || shelf[callNum] == null) {
            System.out.println("Book nonexistant");
            return;
        }
        if (shelf[callNum].user != null) {
            System.out.println("Book already checked out.");
            return;
        }
        shelf[callNum].daysTillDue =
            DocsIO.readint("How many days until due? ");
        shelf[callNum].user = membership[patronId];
        membership[patronId].checkedOut[checkoutNumber] =
            shelf[callNum];
    }
```

Library example

```
static void returnBook(Patron[] membership) {
    int patronId = DocsIO.readint("Patron id? ");
    if (patronId < 0 || patronId > membership.length ||
        membership[patronId] == null) {
        System.out.println("Patron nonexistant");
        return;
    }
    int callNum = DocsIO.readint("Book call number? ");
}
```

Library example

```
 int checkoutNumber = -1;
for (int i = 0; i < membership[patronId].checkedOut.length; i++)
    if (membership[patronId].checkedOut[i] != null &&
        membership[patronId].checkedOut[i].callNumber == callNum) {
        checkoutNumber = i;
        break;
    }
if (checkoutNumber == -1) {
    System.out.println("No such book checked out.");
    return;
}
```

Library example

```
    if (membership[patronId].checkedOut[checkoutNumber].daysTillDue < 0)
        membership[patronId].fines +=
            50 * (0 -
            membership[patronId].checkedOut[checkoutNumber].daysTillDue);
    membership[patronId].checkedOut[checkoutNumber].user = null;
    membership[patronId].checkedOut[checkoutNumber] = null;
}
```

Library example

```
static void payFine(Patron[] membership) {
   int patronId = DocsIO.readint("Patron id? ");
   if (patronId < 0 || patronId > membership.length ||
       membership[patronId] == null) {
       System.out.println("Patron nonexistant");
       return;
   }
   System.out.println(membership[patronId].name + " owes "
                      + membership[patronId].fines + " cents");
   int payment = DocsIO.readint("Amount paid? ");
   membership[patronId].fines -= payment;
}
```

Library example

```
static void payFine(Patron[] membership) {
   int patronId = DocsIO.readint("Patron id? ");
   if (patronId < 0 || patronId > membership.length ||
       membership[patronId] == null) {
       System.out.println("Patron nonexistant");
       return;
   }
   System.out.println(membership[patronId].name + " owes "
                      + membership[patronId].fines + " cents");
   int payment = DocsIO.readint("Amount paid? ");
   membership[patronId].fines -= payment;
}
```

Library example

```
static void payFine(Patron[] membership) {
   int patronId = DocsIO.readint("Patron id? ");
   if (patronId < 0 || patronId > membership.length ||
       membership[patronId] == null) {
       System.out.println("Patron nonexistant");
       return;
   }
   System.out.println(membership[patronId].name + " owes "
                      + membership[patronId].fines + " cents");
   int payment = DocsIO.readint("Amount paid? ");
   membership[patronId].fines -= payment;
}
```

Library example

```
static void payFine(Patron[] membership) {
   int patronId = DocsIO.readint("Patron id? ");
   if (patronId < 0 || patronId > membership.length ||
       membership[patronId] == null) {
       System.out.println("Patron nonexistant");
       return;
   }
   System.out.println(membership[patronId].name + " owes "
                      + membership[patronId].fines + " cents");
   int payment = DocsIO.readint("Amount paid? ");
   membership[patronId].fines -= payment;
}
```

Library example

```
    static void newDay(Book[] shelf) {
        for (int i = 0; i < shelf.length; i++)
            if (shelf[i] != null && shelf[i].user != null)
                shelf[i].daysTillDue -= 1;
    }
}
```