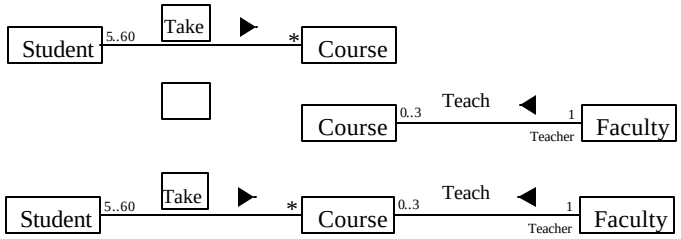
Relationships among Classes

- Association
- Aggregation
- Inheritance

Association

Association represents a general binary relationship that describes an activity between two classes.

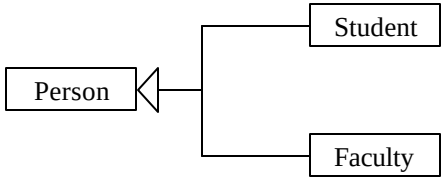


Aggregation

Aggregation is a special form of association, which represents an ownership relationship between two classes. Aggregation models the relationship like has-a, part-of, owns, and employed-by.

Inheritance

Inheritance models the is-a relationship between two classes.

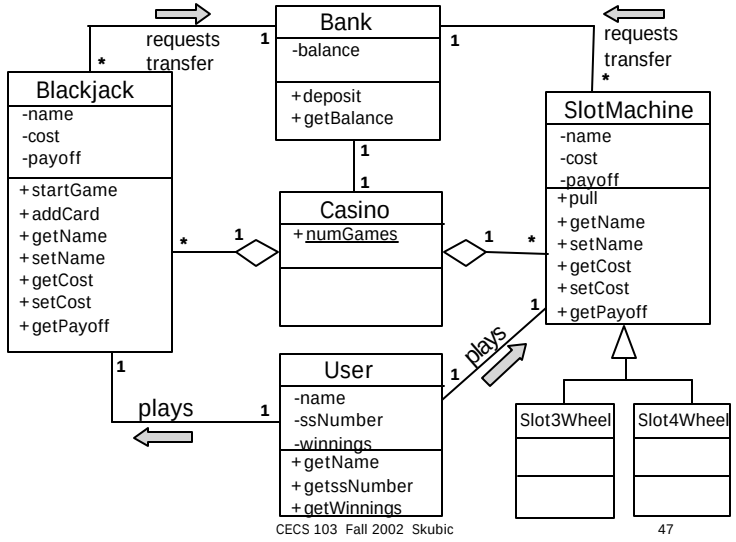


Class Abstraction

Class abstraction means to separate class implementation from the use of the class. The creator of the class provides a description of the class and lets the user know how the class can be used. The user of the class does not need to know how the class is implemented. *The detail of implementation is encapsulated and hidden from the user.*

Class Design

1. Identify classes for the system.
2. Describe attributes and methods in each class.
3. Establish relationships among classes.
4. Create classes.



Low Rollers Casino: Which Slot Machine is the best?

Input line: machineName cost-to-play low high

Use **Random class, seed=1** and the payoff table to "play" the machine 50 times and compute the money won or lost

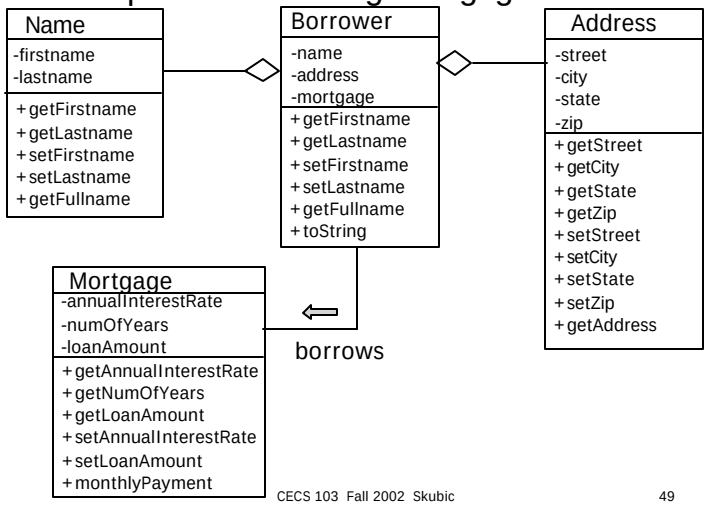
Example file:

LuckyDucky	5	3	10
SpruceGoose	2	1	5
BubbaGump	10	2	7

Combination	1	2	3	4	5	6	7	8	9	10
Payoff	$12	$4	$54	$2	$20	$17	$6	$20	$12	$11

3 classes: Assign4 InputReader SlotMachine

Example 5.6 Borrowing Mortgages



Java API and Core Java classes

- **`java.lang`**

  Contains core Java classes, such as numeric classes, strings, and objects. *This package is implicitly imported to every Java program.*

- **`java.applet`**

  Contains classes for supporting applets.

Java API and Core Java classes, cont.

- **`java.io`**

  Contains classes for input and output streams and files.

- **`java.util`**

  Contains many utilities, such as date, stacks, Random.

- **`java.net`**

  Contains classes for supporting network communications.

Java API and Core Java classes, cont.

- **`java.awt`**

  Contains classes for graphics.

- **`java.awt.image`**

  Contains classes for managing bitmap images.

- **`java.awt.peer`**

  Platform-specific GUI implementation.

The **`String`** Class

- Declaring a String:

```
String msg1 = "Welcome to Java!“
String msg2 = new String("Welcome to Java!“);
String msg3 = new String();
```

- String Comparisons
- String Concatenation
- Substrings
- String Length
- Retrieving Individual Characters in a String

String Comparisons

- **equals**

```
String s1 = "Welcome";
String s2 = "welcome";

if (s1.equals(s2))
{  // s1 and s2 have the same contents  }

if (s1 == s2)
{
 // s1 and s2 have the same reference
}
```


Substrings

The characters in a **`String`** cannot be changed individually.

```
String s1 = "Welcome to Java";
String s2 = s1.substring(0,11) + "HTML";
```

NOW: `s2 = “Welcome to HTML”`

String Concatenation

```
s1 = “Welcome to ”;
s2 = “my world”;
String s3 = s1.concat(s2);
```

NOW: `s3 = “Welcome to my world”`

same as:

```
String s3 = s1 + s2;
```

e.g., `System.out.println(s1+s2);`

Finding String Length

Finding string length using the **`length()`** method:

```
message = "Welcome";
message.length()  // returns 7
```


Retrieving Individual Characters in a String

Do not use **`message[0]`**

Use **`message.charAt(index)`**

*Index starts from **`0`***

Example:

```
String message = "Welcome to my world";
System.out.println(message.charAt(3));
// prints "c"
```