CS133: Developing Programming Principles

Lecture 2

`Board`, `String`, `Scanner` classes
Interactive Input/Output

What is an object?

Rough definition:

- An **object** is an entity that **stores** information about itself (in attributes) and has **methods** to modify/express the stored information.

Class

- Objects of the same type are said to belong to the same **class**.

OR

- Classes **specify** what data and what operations objects of its type possess

A class specifies attributes and methods.
An object has the data to operate on.

Method

- Actions an object can perform
- Examples:

  `String` class

  get length
  look for a character
  convert to upper case
  determine equality

These methods may produce output, compare objects, modify attributes, etc.

Calling a method

```
System.out.println("Message");
```

The **`Board`** class

- An object that displays a graphical, rectangular game board which can be interfaced with and manipulated.
- Operations:
  - adding/removing pegs
  - getting the board size
  - getting input from the user
  - drawing lines, displaying simple messages
- Note: There are no query methods for pegs/lines.

Initializing **`Board`**s

- To initialize:

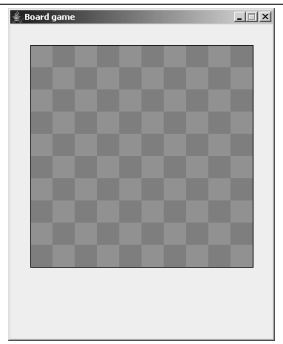
```
Board Name = new Board(int Columns);
```

or

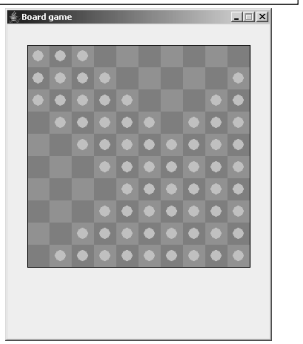
```
Board Name = new Board(int Rows,
                       int Columns);

Board line = new Board(15);
Board grid = new Board(6, 7);
```


The **`Board`** object in action



Empty **`Board`** upon initialization



`Board` with some pegs placed on it

Putting pegs on 1D **`Board`**s

`void putPeg(Color colour, int position)`

places a peg of the specified colour in the given position on a board with just one row.

Example:

```
line.putPeg(Board.BLUE, 4);
```


Putting pegs on 2D **`Board`**s

`void putPeg(Color colour, int row, int col)`

places a peg of the specified colour in the given row and column.

Example:

```
grid.putPeg(Board.YELLOW, 2, 4);
```

Note: You must specify valid row and column.

Removing pegs from 1D **`Board`**s

`void removePeg(int position)`

removes a peg from the specified position on a board with just one row.

Example:

```
line.removePeg(7);
```


Removing pegs from 2D **`Board`**s

`void removePeg(int row, int col)`

removes a peg from the specified location on the board.

Example:

```
grid.removePeg(3, 4);
```

Getting **`Board`** dimensions

`int getColumns()`

Gets size (1D boards)

Gets number of columns (2D boards)

`int getRows()`

Gets number of rows (2D boards)

Examples:

```
int length = line.getColumns();
int numRows = grid.getRows();
int numCols = grid.getColumns();
```


Getting input (1D **`Board`**s)

`int getPosition()`

Waits for the user to click on a position on a board with just one row and then returns it.

Example:

```
int lastClicked =
          line.getPosition();
System.out.println("Last click at"
          + lastClicked);
```

Getting input (2D **`Board`**s)

`Coordinate getClick()`

Waits for the user to click on a position and returns an object containing the row and column.

Example:

```
Coordinate lastClicked =
    grid.getClick();
int r = lastClicked.getRow();
int c = lastClicked.getCol();
System.out.println("Last click at"
    + "(" + r + ", " + c + ")" );
```


More about **`String`**

`String` is a special class in Java:

- **`String`** objects are constructed differently
- Operators (e.g., +) are overloaded
- **`String`**s are **immutable**

But `Strings` have methods just as other classes do.

Basic methods of **`String`**

- **`int length()`**
- **`String toLowerCase()`**
- **`String toUpperCase()`**
- **`char charAt(int position)`**
- **`String substring(int start)`**
- **`String substring (int start, int last)`**
- **`int indexOf(char c)`**

  See p. 80-82 of text for more detail

Overloading

Notice, two methods

`s.substring(int start)`

and

`s.substring(int start, int last)`

have the **same** name!

But their arguments are different in terms of:

- Number and/or
- Type

Therefore, Java knows these are different.
This is called **overloading** a method.

Escape characters

- Used to input special characters into **`String`**s. For example:

```
String hello = "Say \"Hi\"";
```

The value of **`hello`** is:

```
Say "Hi"
```


Escape characters continued

- Useful escape characters:
  - **`\"`** Double quote
  - **`\'`** Single quote
  - **`\\`** Backslash
  - **`\n`** New line
  - **`\t`** Tab

Screen output

- **`System.out.println(`*output*`)`**

  goes to the next line after printing the output on the screen.

- **`System.out.print(`*output*`)`**

  stays on the same line after printing the output on the screen.

The **`Scanner`** class

- Used for keyboard input in Java.
- Initialization:

  ```
  import java.util.*;
  ```

  (as the first line of the program or in the interactions pane)

  ```
  Scanner myScanner =
      new Scanner(System.in);
  ```

Delimiter

- A character or set of characters that is used to separate input.
- Whitespace (space, tab, new line character) by default.

Reading tokens

- **`String next()`**

  reads all characters up to a delimiter into a **`String`**. Skips over delimiters at the beginning.

  Example:

```
String myInput =
    myScanner.next();
```

Reading lines

- **`String nextLine()`**

  reads all characters up to the end of the line into a **`String`**. Includes all delimiters at beginning and before end of line.

  Example:

```
String myInput =
    myScanner.nextLine();
```


Reading integers

- **`int nextInt()`**

  reads the next value of type **`int`** that is entered at the keyboard.

  Example:

```
int myInteger =
    myScanner.nextInt();
```

For a full list of methods, refer to p.92-93

Other methods

- **`boolean hasInt()`**
- **`boolean hasNext()`**

  Return true if input of the specified type is available.

  Useful for error checking.

  We don't officially use these in the course.

For a full list of methods, refer to p.92-93 of the text.

Using the **`Scanner`** class

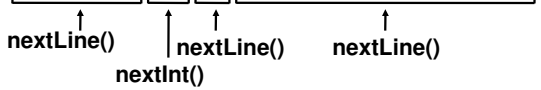
```
Scanner myScanner = new Scanner(System.in);
System.out.println("Enter your name:");
String name = myScanner.nextLine();
System.out.println("Enter your age:");
int age = myScanner.nextInt();
System.out.println("Enter your address:");
String address = myScanner.nextLine();
```

Using the **`Scanner`** class continued

- Sample input:

```
Troy
42
This is now!
```

- The actual input **`String`** is:



ERROR: undefined
OFFENDING COMMAND:

STACK: