CS133: Developing Programming Principles

Lecture 4

Loops, `static` methods, `Math` class

`while` loop

- Syntax:

```
while (Boolean_Expression)
{
   Body_Statements;
}
```

`while` loop continued

- Example:

```
// Assume number is given
int steps = 0;
while (number > 1 && steps <= 200)
{
   if (number % 2 == 0) {
      number = number/2;
   } else {
      number = number*3+1;
   }
  steps++;
}
```


`for` loop

- Syntax:

```
for ( Initializing_Action;
      Boolean_Expression;
      Update_Action)
{
   Body_Statements;
}
```

`for` loop continued

- Example:

```
for (int i = 5; i >= 0; i--)
{
  System.out.println(i);
}
System.out.println("Blast off!");
```


Problem

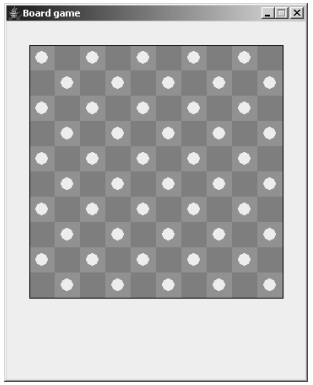
- Draw a square of asterisk (`*`) characters for a given number of rows/columns.
- Sample output (num = 4):

```
****
****
****
****
```

Making patterns

```
// Places yellow pegs on the board in a
// checker pattern
Board myBoard = new Board (10, 10);
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 10; j++) {
    if ((i+j) % 2 == 0) {
      myBoard.putPeg(Board.YELLOW, i, j);
    }
  }
}
```


Making patterns continued



Tracing variables

- Observing the change in value of variables through different iterations of a loop.
- Allows us to spot abnormalities and fix otherwise elusive bugs.
- DrJava allows us to trace our Java code.

`break` statement

- **`break;`**

  allows us to stop a loop from executing before the loop condition is met.

  Stops executing immediately.
- It is difficult to keep track of all possible outcomes.
- Use it sparingly!

do-while loop

- Syntax:

```
do
{
   Body_Statements;
} while (Boolean_Expression);
```


do-while loop continued

- Example:

```
double price = 9000;
do
{
  price *= 0.10;
} while (price >= 100.0);
```

Guessing game

```
Scanner input = new Scanner(System.in);
String guess;

do {
  System.out.println("Guess my password:");
  guess = input.nextLine();
} while (!guess.equals("Password"));

System.out.println("You got it!");
```


Exiting a program

- **`System.exit(0);`**

  will exit the program at any point it is invoked.

  Usually unnecessary but may be helpful when developing programs.

The **`Math`** class

- A predefined class of static methods for common math functions:

Name	Description	Type of argument	Type of value returned
`random`	Generates 0<=number<1	none	`double`
`abs`	Absolute value	`int`, `long`, `float`, `double`	Same as type of argument
`max`	Maximum	`int`, `long`, `float`, `double`	Same as type of argument
`min`	Minimum	`int`, `long`, `float`, `double`	Same as type of argument
`sqrt`	Square root	`double`	`double`

Using **`Math.random()`**

- Generate an integer between A and B (A ≤ B):

```
int myNumber =
(int)(Math.random()*(B-A+1) + A);
```

Example:

```
int number =
        (int)(Math.random()*9 + 7);
// Generates a random number between
// 15 and 7
```

`static` methods

- Can be accessed without instantiating an object of the class type.
  - For now: use only for **`Math`** functions
  - We will learn more about **`static`** later
- Syntax:

  ***ClassName*.*Method*(*Arguments*);**

ERROR: undefined
OFFENDING COMMAND:

STACK: