# CSC 222: Object-Oriented Programming

# Fall 2017

## Simulations & library classes

- HW3: RouletteWheel, RouletteGame, RouletteTester
- javadoc
- java.lang classes: String, Character, Integer
- java.util.Random
- for vs. while loops
- scope and static fields/methods



# Simulations

## programs are often used to model real-world systems

- often simpler/cheaper to study a model
- easier to experiment, by varying parameters and observing the results

- Pig game simulation allowed us to compare strategies across a statistically significant number of games

## HW3: you will use & develop classes that simulate different betting strategies for roulette

- if you start with 100 credits and play 100 spins of the wheel, what is your best strategy for winning?

- how do we define winning? ending the rounds with a profit? ending the rounds with the most credits?

## Spam alert

Re: hi bud

[redacted]

To: David Reed <davereed@creighton.edu>

yo mate, ok I`ll give you my trick so you can get some extra cash on xmas :) but if you give it someone else I`ll f[redacted]in kill you :)
you know in roulette you can bet on blacks or reds. If you bet $1 on black and it goes black you win $1 but if it goes red you loose your $1.
So I found a way you can win everytime:

bet $1 on black if it goes black you win $1

now again bet $1 on black, if it goes red bet $3 on black, if it goes red again bet $8 on black, if red again bet $20 on black, red again bet $52 on black (always multiple you previous lost bet around 2.5), if now is black you win $52 so you have $104 and you bet:

$1 + $3 + $8 + $20 + $52 = $84 So you just won $20 :)

now when you won you start with $1 on blacks again etc etc. its always bound to go black eventually (it`s 50/50) so that way you eventually always win. But there`s a catch. If you start winning too much (like $1000 a day) casino will finally notice something and can ban you. I was banned once on royal casino. So don`t be too greedy and don`t win more then $200 a day and you can do it for years. I think bigger casinos know that trick so I play for real money on smaller ones, right now I play on [redacted]: www.[redacted] for more then 3 months, I win $50-$200 a day and my account still works. You`ll find roulette there when you log in go to "specialty" section - "american roulette". And don`t you dare talling about it anyone else, if too many people knows about it casinos will finally found a way to block that trick. If you have any questions just drop me a line here or on skype.

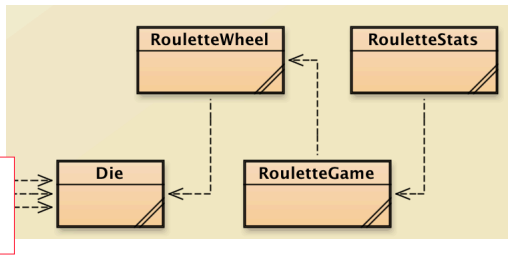
c ya



## HW3 classes

you will write parts of a project that includes several interacting classes

- **RouletteWheel:** models a roulette wheel that can be spun, access the number, color or parity of the spin
- **RouletteGame**: uses RouletteWheel to simulate a game with betting, keeping track of winnings/losses
- **RouletteStats**: uses RouletteGame to perform repeated simulations and display stats (can be used to compare betting strategies)



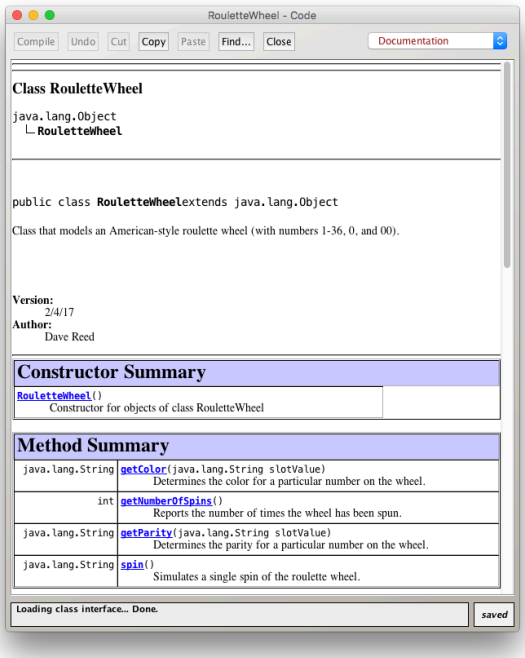

recall: Die is used by lots of other classes

# Abstraction & classes

note that RouletteGame will depend upon RouletteWheel

- but you don't have to know the details of how that class works
- can abstract away the details and focus on its behavior

- recall that javadoc comments can be used to document the behavior of a class
- always include javadoc comments for each class (incl. `@author` & `@version`) and for every method (incl. `@param` & `@return`)

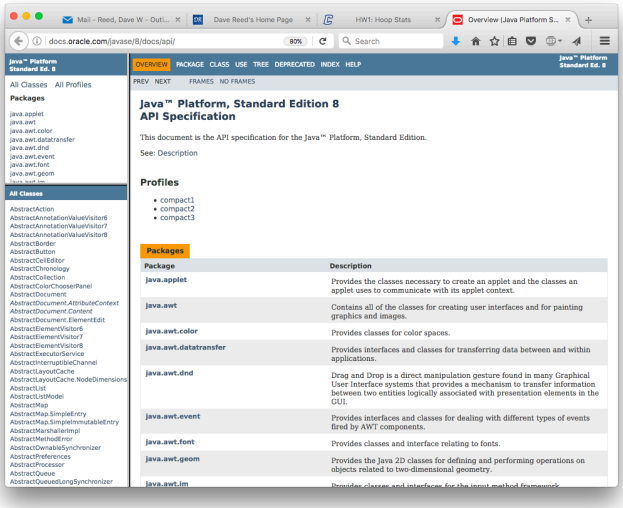



5

# Java Standard Library

the Java language provides an extensive library of nearly 4,000 classes

- documentation for the entire library is accessible in javadoc form
- to view in BlueJ, select Java Class Libraries under the Help menu




6

# String class

## one of the most useful library classes is String

- technically, its full name is `java.lang.String`, but the `java.lang` prefix can be omitted
- a String object encapsulates a sequence of characters and many useful methods
- since Strings are so common, Java provides shortcuts to make them easier to use (and look like primitives)

```
String str = "foo";  ➔  String str = new String("foo");
```

- we have already seen a few basic operations on Strings
  you can display Strings using `System.out.print` and `System.out.println`

```
System.out.println(firstName);
```

the '+' operator concatenates two strings (or string and number) together

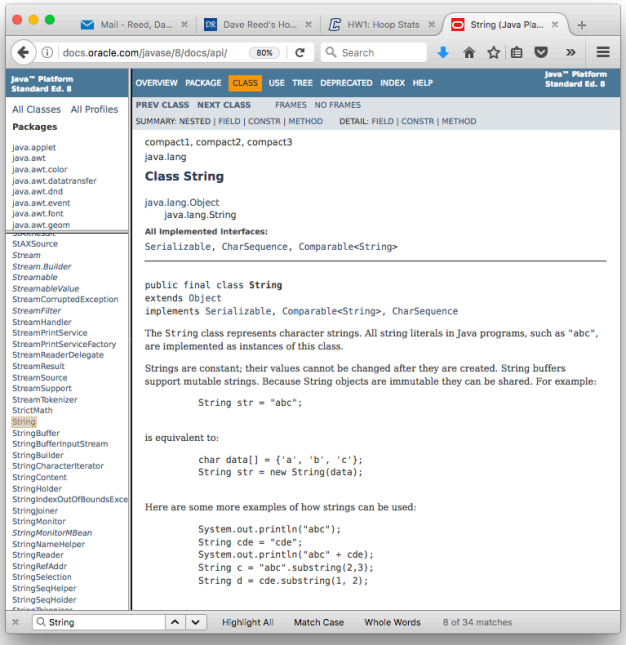
```
String str = "foo" + "lish" + 1;
```

7

# String javadoc

## many of the javadoc details will be revealed later

- for now, important feature is that we can scan the constructors & methods of a class
- String has MANY of each
- can click on a constructor/method link to see more details




8

## Common String methods

| | |
|---|---|
| `int length()` | returns number of chars in String |
| `char charAt(int index)` | returns the character at the specified index (indices range from 0 to str.length()-1) |
| `boolean contains(String str)` | returns true if str occurs in the String, else false |
| `int indexOf(char ch)` | returns index where ch/str first occurs in the String (-1 if not found) |
| `int indexOf(String str)` | |
| `String substring(int start, int end)` | returns the substring from indices start to (end-1) |
| `String toUpperCase()` | returns copy of String with all letters uppercase |
| `String toLowerCase()` | returns copy of String with all letters lowercase |
| `boolean equals(String other)` | returns true if other String has same value |
| `int compareTo(String other)` | returns -1 if less than other String,<br>0 if equal to other String,<br>1 if greater than other String |

9

## Character class

recall: in Java, strings and characters are different types

`String` is a class, `char` is a primitive type

the (`java.lang.`)`Character` class has numerous static methods for manipulating characters

| | |
|---|---|
| `char toLowerCase(char ch)` | returns lowercase copy of ch |
| `char toUpperCase(char ch)` | returns uppercase copy of ch |
| `boolean isLetter(char ch)` | returns true if ch is a letter |
| `boolean isLowerCase(char ch)` | returns true if lowercase letter |
| `boolean isUpperCase(char ch)` | returns true if uppercase letter |

10

## String/Character examples

```
public boolean isVowel(char ch) {
    String vowels = "aeiouAEIOU";
    return vowels.contains(""+ch);
}


public char randomChar(String str) {
    Die d = new Die(str.length());
    return str.charAt(d.roll()-1);
}


public String capitalize(String str) {
    if (str.length() == 0) {
        return str;
    }
    else {
        return Character.toUpperCase(str.charAt(0)) +
               str.substring(1, str.length());
    }
}
```

11

## Comparing strings

comparison operators (< <= > >=) are defined for primitives but not objects

```
String str1 = "foo", str2 = "bar";
if (str1 < str2) …                  // ILLEGAL
```

== and != are defined for objects, but don't do what you think

```
if (str1 == str2) …                 // TESTS WHETHER THEY ARE THE
                                    // SAME OBJECT, NOT WHETHER THEY
                                    // HAVE THE SAME VALUE!
```

Strings are comparable using the `equals` and `compareTo` methods

```
if (str1.equals(str2)) …            // true IF THEY REPRESENT THE
                                    // SAME STRING VALUE

if (str1.compareTo(str2) < 0) …     // RETURNS neg # if str1 < str2
                                    // RETURNS   0   if str1 == str2
                                    // RETURNS pos # if str1 > str2
```

12

## Comparison example

suppose we wanted to compare two names to see which comes first alphabetically

- Kelly Jones < Kelly Miller < Chris Smith < Pat Smith

```
public void compareNames(String myFirst, String myLast,
                         String yourFirst, String yourLast) {
    int lastCompare = myLast.compareTo(yourLast);
    int firstCompare = myFirst.compareTo(yourFirst);

    if (lastCompare < 0 || (lastCompare == 0 && firstCompare < 0)) {
        System.out.println("My name comes before yours alphabetically!");
    }
    else if (lastCompare > 0 || (lastCompare == 0 && firstCompare > 0)) {
        System.out.println("Your name comes before mine alphabetically!");
    }
    else {
        System.out.println("We have the same name!");
    }
}
```

note: we have been using == to compare Strings up to this point

- dangerous – sometimes it works, sometimes it doesn't!
- from now on, always use `.equals` for Strings

13

## RouletteWheel implementation

- we can use a Die object to choose between 38 values
- the spin method must return a String, since we may want to differentiate between 0 and 00
- 38 → "00", 37 → "0"
- 1-36 are converted to a string by concatenating with the empty string
  e.g., ""+36 → ""+"36" → "36"
- the Die field already keeps track of the number of rolls/spins

```
public class RouletteWheel {
  private Die roller;

  public RouletteWheel() {
    this.roller = new Die(38);
  }

  public String spin() {
    int number = this.roller.roll();
    if (number == 38) {
      return "00";
    }
    else if (number == 37) {
      return "0";
    }
    else {
      return ""+number;
    }
  }

  public int getNumberOfSpins() {
    return this.roller.getNumRolls();
  }

  public String getColor(String slotValue) {
    // NEXT SLIDE
  }

  public String getParity(String slotValue) {
    // SLIDE AFTER THAT
  }
}
```

14

## RouletteWheel implementation (cont.)

- getting the color associated with a number is not obvious
  we could have a large cascading if else to map each number to a color
- instead can use the String method `contains`
  `redNums` & `blackNums` contain all the numbers of each color, with spaces
  to see if `"3"` is red/black, see if `redNums`/`blackNums` contains `" 3 "`

```
public class RouletteWheel {
  . . .

  public String getColor(String slotValue) {
    String redNums = " 1 3 5 7 9 12 14 16 18 19 21 23 25 27 30 32 34 36 ";
    String blackNums = " 2 4 6 8 10 11 13 15 17 20 22 24 26 28 29 31 33 35 ";

    if (redNums.contains(" "+slotValue+" ")) {
      return "red";
    }
    else if (blackNums.contains(" "+slotValue+" ")) {
      return "black";
    }
    else {
      return "green";
    }
  }
}
```

15

## RouletteWheel implementation (cont.)

- similarly, could have a large cascading if else to map each number to odd/even
- or, could use `charAt` to access the last digit, see if it is "0", "2", "4", "6", or "8"
- instead, the (`java.lang.`)`Integer` class contains a static method for converting a string of digits into an int: `parseInt`
  e.g., `Integer.parseInt("14")` → `14`

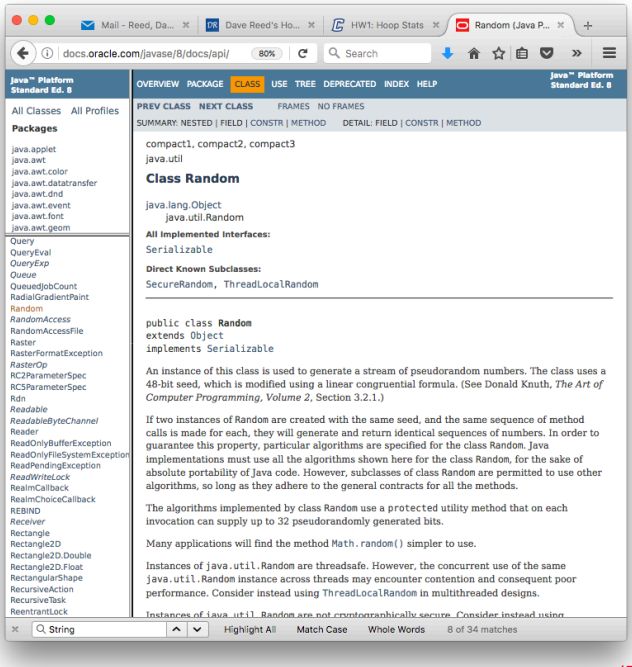
```
public class RouletteWheel {
  . . .

  public String getParity(String slotValue) {
    int numVal = Integer.parseInt(slotValue);
    if (numVal == 0) {
      return "zero";
    }
    else if (numVal % 2 == 0) {
      return "even";
    }
    else {
      return "odd";
    }
  }
}
```

16

# Random class

instead of the Die class, we could have used the `java.util.Random` class

- the nextInt method returns a random int from 0..(parameter-1)
- other methods can be used to generate different random numbers




17

# RouletteWheel w/ Random

- classes defined within the `java.lang` library are automatically available

  `java.lang.Math`
  `java.lang.System`
  `java.lang.String`
  `java.lang.Character`
  `java.lang.Integer`

- we won't list the unnecessary prefix (but Javadoc will)
- for other libraries, must explicitly `import` the class

```
import java.util.Random;

public class RouletteWheel {
  private Random randGen;

  public RouletteWheel() {
    this.randGen = new Random();
  }

  public String spin() {
    int number = this.randGen.nextInt(38);
    if (number == 37) {
      return "00";
    }
    else {
      return ""+number;
    }
  }

  public int getNumberOfSpins() {
    // SAME AS BEFORE
  }

  public String getColor(String slotValue) {
    // SAME AS BEFORE
  }

  public String getParity(String slotValue) {
    // SAME AS BEFORE
  }
}
```

18

## Variable scope

*scope:* the section of code in which a variable exists

- for a field, the scope is the entire class definition
- for a parameter, the scope is the entire method
- for a local variable, the scope begins with its declaration & ends at the end of the enclosing block (i.e., right curly brace)

```
public class DiceStuff {
  private Die d6;

  . . .

  public void showSevens(int numReps) {
    int count = 0;
    for (int numRolls = 0; numRolls < numReps; numRolls++) {
      if (d6.roll() + d6.roll() == 7) {
        count++;
      }
    }
    System.out.println(count);
  }

  . . .
}
```

since each method defines its own scope, can reuse the same parameter/local variable name in multiple methods

within a method, can have a parameter/local variable with same name as a field (although confusing)

use `this.` to differentiate

19

## Static fields & constants

### static fields

- a field declared to be static is shared by all objects of that class
- useful when there is data that needs to be accessed/updated by all objects
- also useful for constants – values that will not change during execution (so there is no reason for object to have its own copy)

```
public class RouletteStats {
  private final static int START_CREDITS = 100;
  private final static int BET_AMOUNT = 1;

  // METHODS CAN ACCESS THESE CONSTANTS
}
```

- note: constants are initialized when declared (not in a constructor)
- convention is to use all caps for constants

- any attempt to reassign a final value will cause a compiler error

20

## Static methods

sometimes a class doesn't require fields (other than constants)

- without fields, every object of the class would look/behave exactly the same
- essentially, it is a collection of independent functions/methods

- so, why even create objects at all?
- the alternative is to declare the methods to be static
  - ✓ that way, they belong to the class
  - ✓ you can call them directly on the class, without creating an object

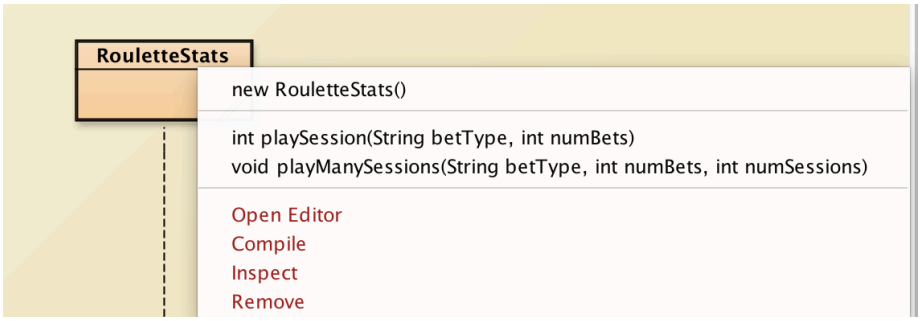
```
public class RouletteStats {
   private final static int START_CREDITS = 100;
   private final static int BET_AMOUNT = 1;

   public static int playSession(String betType, int numBets) {
      …
   }

   public static void playManySessions(String betType, int numBets,
                                       int numSessions) {
      …
   }
}
```

21

## Static methods in BlueJ

when you have a class with only static methods,

- BlueJ will still list a default constructor (which creates a default object)
- will also see a list of static methods
- you don't need to create an object, just call the methods directly




22