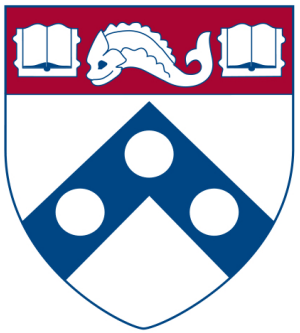


Penn Engineering

Online Learning

Video 3.1

Arvind Bhusnurmath

ArrayList

- Used when the size of the data collection is unknown.
- The indexing operations are still quick
- Adding an element is quick on average.
  - ArrayList works by dynamically resizing an array behind the scenes.

Syntax for ArrayList

- Remember to import java.util.*
- ArrayList<object datatype>;
- An arraylist can only be made of objects. Primitive datatypes are not allowed.

ArrayList of Strings

- Declaration

```
ArrayList<String> names;
```

- Initialize the list

```
names = new ArrayList<String>();
```

- Declare and initialize

```
ArrayList<String> names =
                        new ArrayList<String>();
```

ArrayList methods

```
names = new ArrayList<String>();
```

- `names.add("John Doe")` - add the element to the end of the arraylist
- `names.get(i)` - get the element at the i^{th} index.
- `names.contains("John Doe")` – returns a boolean saying whether the names arraylist contains John Doe
- `names.remove(i)` – remove the element at the i^{th} index.

Java wrapper classes for primitives

- Java has classes that ”wrap around” the primitive datatypes.
- Instead of using the primitive datatype int you can replace it with java’s Integer class.
- To make an arraylist containing integers (in the mathematical sense)

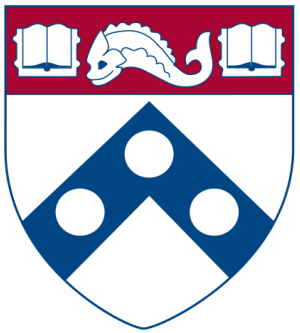
```
ArrayList<Integer> attendance =
      new ArrayList<Integer>();
```

Example – using ArrayList to simulate cards

```
public Card {
    char suit;// d, s, c, h
    int rank; // Jack is 11, Queen is 12, King is 13
}
//creating a deck of cards
ArrayList<Card> deck = new ArrayList<Card>();
suits = new char[] {'d', 's', 'c', 'h'};
for (int i = 1; i < = 13; i++) {
   Card c = new Card();
   for (int j = 0; j <= 4; j++) {
      c.suit = suits[j]; c.rank = i;
      deck.add(c);
   }
}
```

Shuffling an ArrayList

```
Collections.shuffle(deck);
//pick the 10th card
Card tenthCard = deck.get(9);
System.out.println(tenthCard.rank + " " +
                                    tenthCard.suit);
```



Penn Engineering

Online Learning

Video 3.2

Arvind Bhusnurmath

Enhanced for loop

- For any collection of data the following syntax loops through every element

```
for( datatype variable : collection) {
      // variable name can be used in this loop
        // variable takes each value in the
       //collection one by one
}
```

Example: find longest name in a list

Assume we have an arraylist of strings called names

```
int max = 0;
String longest = “”;
for (String name : names) {
    int current = name.length();
    if (current > max) {
        max = current;
        longest = name;
    }
}
```

Looping through arrays

The enhanced for loop can work for arrays as well

```
int[] scores = //an array of integer scores;
double total = 0;

for (double element : scores) {
  total = total + element;
}
```

Modifying an array in a loop

- The enhanced for loop is best used for reading elements
- Not suitable for initializing values or modifying existing values

```
for (double element : values) {
    element = 0;
}
```

The above loop does not make all the values in the collection called values to be 0.

Modifying an array(the corrent way)

```
for (int i = 0; i < values.length; i++) {
    values[i] = 0;
}
```

Modifying an arraylist

```
for (int i = 0; i < values.size(); i++) {
    values.set(i, 0);
}
```

ConcurrentModificationException

```
for (Card ca : deck) {
    if (ca.suit == 'c') {
            deck.remove(ca);
       }
}
```

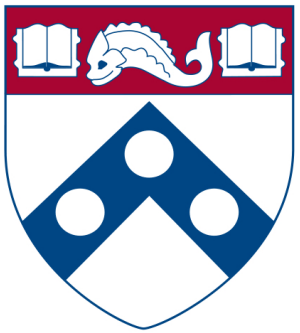
- Removing elements while iterating through the list at the same time is not allowed in the enhanced for loop.

Solving the ConcurrentModificationException

```
ArrayList<Card> clubs = new ArrayList<Card>();

for (int i = 0 ; i < deck.size(); i++) {
   Card ca = deck.get(i);
   if (ca.suit == 'c') {
      clubs.add(ca);
   }
}

deck.removeAll(clubs);
```



Penn Engineering

Online Learning

Video 3.3

Arvind Bhusnurmath

Documentation

- In order to get someone to use the classes that you develop it is important to communicate to them
- No one wants to read every single line of your code
- The best way is to have good documentation

Javadoc creation

- Provide documentation for every method
- In Eclipse just type in /** before a method and hit enter

```
public int getNumerator() {
    return 5;
}
```

Javadoc creation

```
/**
public int getNumerator() {
    return 5;
}
```

Javadoc creation

```
/**
 *
 * @return
 */
public int getNumerator() {
    return 5;
}
```

Tags in documentation

- Eclipse auto generates a block of documentation for you
- Fill in general documentation about what the method does. Remember that there are tags to provide more detail about components of the method.
- @param - provide helpful documentation for each parameter
- @return – clearly specify what the method returns
- One additional tag is
  - @see – if you want the reader of your documentation to look up a different class

Using Eclipse to generate documentation

- Project - > Generate Javadoc
- Remember to write your javadocs completely before clicking this
  - every public method must have some javadocs.

Using Eclipse to generate documentation

```
public class Rational {

    int num; // the numerator
    int den; // the denominator

    /**
     * Given a numerator and a denominator
     * make a rational number of the form
     * <br>
     * numerator
     * <br>
     * ---------
     * <br>
     * denominator.
     *
     * <br>
     * Initialize the rational number.
     * This throws an ArithmeticException if the denominator is 0.
     * @param numerator - the numerator of the fraction
     * @param denominator - the denominator of the fraction.
     */
    public Rational(int numerator, int denominator) {
```

Using Eclipse to generate documentation

Constructor Detail

Rational

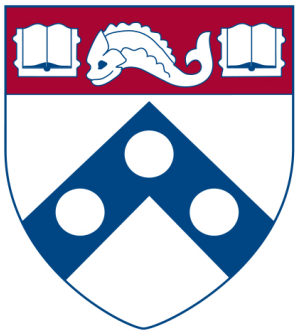
```
public Rational(int numerator,
                int denominator)
```

Given a numerator and a denominator make a rational number of the form
numerator
---
denominator.
Initialize the rational number. This throws an ArithmeticException if the denominator is 0.

Parameters:

`numerator - - the numerator of the fraction`

`denominator - - the denominator of the fraction.`



Penn Engineering

Online Learning

Video 3.4

Arvind Bhusnurmath

Some of the slides in this deck were reproduced with the permission of Dr. David Matuszek.

Topics

- What does static mean?
- Static instance variables
- Static methods

What does static mean?

Think of static as belonging to the class and not to the individual instances of the class.

A static method means the method exists at the class level and is not specific to the instance.

The one static method that you have in a lot of your classes is **main.**

Used when you do not need an instance of the object.

Math.sqrt is a method that computes the square root of a number. You do not need an instance of mathematics before you know how to compute the square root.

Using static methods

- To use a static method called method1 in a class called Class1 the code is
  `Class1.method1(parameters…)`
- You do not have to create an instance of the class in order to invoke the method
- Many java utility functions are static methods. Almost all math function are static methods in the **Math** class.

Static versus non static

Consider the Rational number class

To reduce 2/4 to 1/2 a common thing to do is to compute the greatest common divisor. To compute the gcd you do not need an instance of a fraction.

```
public static int gcd (int a, int b)
public Rational add(Rational otherRational)
```

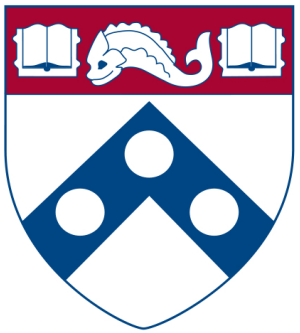
Static instance variables

- Static instance variables are used if you want some information to be shared by every instance of a class
- A common use case is constants
- For constants you also get to see the keyword **final.** Final meaning you do not get to override this value in any manner.
- `public static final int MAXSCORE = 100;`

Example of static variable used to keep count

- Bank account and bank account numbering

- Demo



Penn Engineering

Online Learning

Video 3.5

Arvind Bhusnurmath

Some of the slides in this deck were reproduced with the permission of Dr. David Matuszek.

Topics

- Access Modifiers
- What is private, public?
- What if we don't write any modifier?

Access modifier

- Every instance variable and every method can be given one of 4 access modifiers
  - public
  - protected
  - private
  - default – no modifier provided at all

- We will discuss what protected means later (after we have covered inheritance)

public instance variables

- With an access modifier being public the instance variable or method can be directly accessed, even outside the class

- In the example below, the Spy class can go and change the id of a Human object because the id has a public access modifier

```
public class Human {
    public int id;
    public String name;
}

public class Spy() {
    public static void main(String[] args) {
        Human h = new Human();
        h.id = 45;
    }
}
```

private instance variables

- For the same example (Human, Spy), if the id is made private then it cannot be accessed in the Spy class.
- It can still be accessed within the Human class

```
public class Human {
    private int id;
    public String name;
    public void sayHello() {
      System.out.println("Hello there " + id); // same class.
Works fine.
    }
}

public class Spy() {
    public static void main(String[] args) {
        Human h = new Human();
        h.id = 45; // this line of code will not work anymore
    }
}
```

Default access modifier

- What happens if you leave out the private/public?
- The default access is for every class inside the same package to be able to access the instance variable while classes in different packages cannot do so.
- This is quite uncommon in the actual software industry. Avoid it unless there is a very specific need.

Best practice for instance variables

- The first preference for any instance variable is to make it private
- You still want to be able to access these instance variables. The correct way of doing so is via accessors and mutators
- Accessors and mutators
  - Also called getters and setters

```
public class Student {
    private String name;
    public getName() {
         return name;
    }
    public setName(String name) {
        this.name = name;
    }
}
```

Best practice for instance variables

```
public class Student {
    private String name;
    public getName() {
         return name;
    }
    public setName(String name) {
        this.name = name;
    }
}

public class School {
  private Student[] students;
  public void printStudentNames() {
     for (int i = 0; i < students.length; i++) {
        System.out.println(students[i].getName());
     }
  }
}
```

public methods

- The method can be accessed from any other class.
- Public methods are the primary manner in which two classes communicate with each other.
- Think of a public method as a service that one class is providing to another.

private methods

- A private method can only be accessed within the class
- Common use case – Helper methods

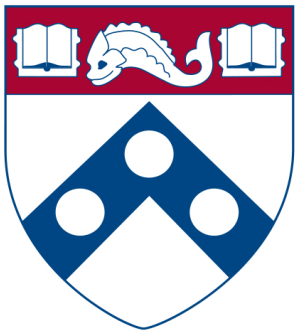
```
public class Student {
    private int age;
    public void setAge(int age) {
        if (verifyAge(age))
            this.age = age;
    }
    private verifyAge(int age) {
         return age >= 1;
    }
}
```

Summary

Modifier	Class	Package	World
public	Y	Y	Y
default	Y	Y	N
private	Y	N	N

https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

World means any class that is outside the package



Penn Engineering

Online Learning

Video 3.6

Arvind Bhusnurmath

Some of the slides in this deck were reproduced with the permission of Dr. David Matuszek.

Topics

- Hands on example of cars being parked in a parking garage.
- We will design a car object and a parking garage object in Eclipse.