Advanced Programming Techniques in Java

Object Oriented Programming

Lecture 10

Class Objectives

- Arrays of objects (second subsection of 7.4
- Method Overloading (last subsection of 3.1

- Encapsulation (Section 8.4)

```
public class Point{
  int x; int y;

  // constructor
  public Point(int initialX, int initialY){
    x = initialX; y = initialY;
  }

  // constructor
  public Point(){
  x = 0; y = 0;
  }

  // shifts points location by the given amount
  public void translate (int dx, int dy){ x +=
  dx; y += dy;
  }

  // computes the distance between two points
  public double distance(Point other){ int dx
  = x - other.x; int dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
} …
```

Review: Point Class (ver. 5)

Point.java

Point.java (cont.)

```
...
   /
  pu
  Po
  }

  pu

  }

  //
  //
  pu
  Po


  }
}
```

Review: Final version of `equals`

- This version of the `equals` method allows us to c any other type of object:

```
// Returns whether o refers to a P
// the same (x, y) coordinates as
object public boolean equals(Objec
instanceof Point) { Point other =
        return x == other.x && y =
    } else { return
        false;
    }
}
```

Review: Template for your equal

```
public boolean equals (Object o){
   if (o instanceof <type>){
          <type> other = (<type>) o;
          //compare the state and re
   } else { return
   false;
   }
}
```

Review: The `this` keyword

- **Definition** The **`this`** keyword refers to the curren
- The `this` keyword is used to eliminate confusion with the same name
  - Refer to a field: `this.`**field**
  - Call a method: `this.`**method**`(`**paramet**
  - One constructor can call another: `this(`**parameters**`);`
- So far, the compiler was converting expressions au
- `x → this.x`
- `setLocation(10,12) → this.setLocation(10,`

Arrays of ob

Arrays of objects

- `String[] words = new String[5];`
- When objects are first constructed their fields are
- `int` are initialized to `0`, `char` to `'0'`, `boolean` to `f`
- Objects are initialized to `null`

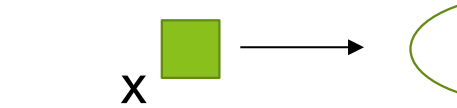
```
int[] numbers= new int[4]; // all ints are
System.out.println(numbers[0]); // prints

String[] words = new String[4]; /all Strin
System.out.println(words[0]); // prints ou
```

null

- Variables declared of a primitive type stores value
- Variables declared of a reference type store refere

```
String x = null
```

```
String x = "abc"
```

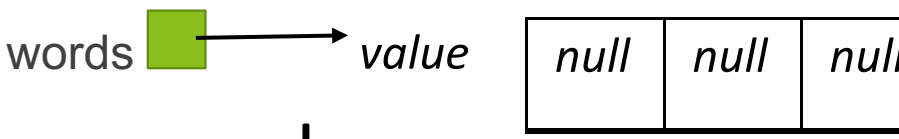


- **Definition** `null` is a value that indicates that the to an object.

  `null`

- The elements of an array of objects are initialized t
- `String[] words = new String[5];`

index	0	1	2



Things you can do

- Store `null` in a variable or an array element
- `String s = null;`
- `words[2] = null;`

- Print a `null` reference
- `System.out.println(s);` **`// output: null`**

- Ask whether a variable or array element is null
- `if (words[2] == null) { …`

- Pass `null` as a parameter to a method
- `System.out.println(null);` **`// null`**
- Return `null` from a method (often to indicate fa

Dereferencing

- Dereferencing happens using the **.** operator

```
String s = "abc";
int x = s.length();    //s is dereferen
```

- Dereferencing follows the memory address place
  where the actual object is located

- When an object has been found the requested method
- If the reference has value `null`, dereferencing results

`NullPointerException`

- It is illegal to dereference `null` (causes an excep
- `null` is not any object, so it has no methods or da

```
String[] words = new String[5];
System.out.println("word is: " + words[0]);
words[0].toUpperCase();
```

words

- **Output** `word is: null`

```
Exception in thread "main" java.lang.NullPoi
at Example.main(Example.java:8)
```

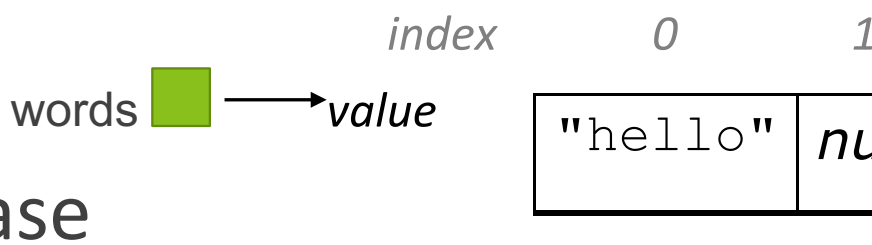
Looking before you leap

- You can check for `null` before calling an object'

```
String[] words = new String[5];
words[0] = "hello";
words[2] = "goodbye";    // words[1], [3], [
```

```
for (int i = 0; i < words.length; i++) { if
    (words[i] != null) { words[i] =
    words[i].toUpperCase();
}
```

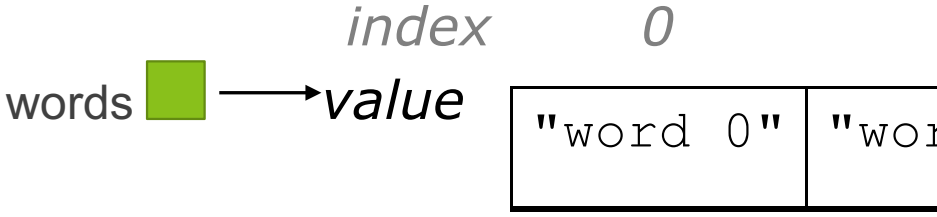
```
}
```



Two-phase initialization

- Initialize the array itself (each element is initially `nu`
- Initialize each element of the array to be a new obj

```
String[] words = new String[4];     // phas
(int i = 0; i < words.length; i++) { words
"word " + i;        // phase 2
}
```



Method Overl

Method Overloading

- Java allows you to **overload a method**
- **Method overloading** is a feature that allows a clas same name but different argument lists
- Note: overloaded methods can only differ in their parame
- **Constructor overloading** allows a class to have m argument lists

Method Overloading

- There are three ways to overload a method
- Number of parameters `add (int, int) add (int, int, int)`
- Data type of parameters `add (int, int) add (int, double)`

- Sequence of data type of parameters `add(int, double) add(double, int)`

Method Overloading: example

- Method 1:

```
public double calcInt(double balance, doub
        return balance * rate;
}
```

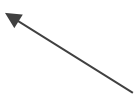
- Method 2:

```
public double calcInt(double balance, int
    double ratePercent = rate/100.0;
                calcInt(1000.00, 4) return b
}
```

Method Overloading: example

- Method 1:

```
public double calcInt(double balance, dou
rate){ return balance * rate;
}
```



Could `calcI` method 1?

- Method 2:

```
public double calcInt(double balance, int
    rate){ double ratePercent = rate/100.
    return balance * ratePercent;
}
```

Method Overloading: example

Compiler recognizes a more exact m
that uses the integer parameter and

- Let's assume we only have this method:

```
public double calcInt(double balance, double rate
        return balance * rate;
}
```

- What happens if you call `calcInt(1000.0, 4)`?
- The method still compiles, and it works (but not correctly
- Compiler will cast 4(integer) to 4.0

- When a data type of smaller size is promoted to th
  this is called type promotion
- Let's assume we have the following methods:

Method Overloading: example

```
public double calcInt(int balance, double
public double calcInt(double balance, int
```

- What happens if you call `calcInt (300, 6)?`
- There is no exact match! Compiler will complain
- **There is always risk when overloading methods programming style (more convenient)**

The `final` keyword for parame

```
public double calcInt(final int balance, dou

public double calcInt(int balance, double ra
```

- `final` keyword means the balance parameter is
  method
- balance is considered a constant within the method `ca`
- These two methods are not overloaded. The comp
- results in compiler/syntax error.

Encapsulat

Encapsulation

- **Definition Encapsulation** refers to the concepts
  an object from the clients of the object

- Protects the integrity of an object's data
- Focusing on the iPod’s external behavior enable u the details of its inner workings

Encapsulation (cont.)

- Encapsulation is a principle of wrapping data (va

- It is one of the four OOP concepts
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Encapsulation example 1

```
public class Account { int
    account_number; int
    account_balance; ...
    public void showData()
    {
        //code to show data
    }
    public void deposit(int a) {
    account_balance = account_balance +
    ...
}
```

- Suppose a hacker managed to gain access to the c
  to deposit amount -100.
- Is that possible?

Encapsulation example 1 (cont.)

```
public class Hacker { public static voi
    main (String[] args){
        ...
        Account a = new Account();
        a.account_balance = -100;
        ...
    }
}
```

- The whole idea behind encapsulation is to hide the

`private` Fields

- To encapsulate the fields of an object, so they cann
  they need to be declared `private`

- **Syntax** `private <type> <name>;`

- If a field is `private` it means it can only be access
- No outside class can access `private` data memb

Back to example 1

```
public class Account { private
    int account_number; private
    int account_balance;

    public void showData() {
        //code to show data
    }

    public void deposit(int a) {
    account_balance = account_balance + a
}
```

- Suppose a hacker managed to gain access to the
tries to deposit amount -100.
- Is that possible?

Back to example 1

```
public class Account {
    private int account_number;
    private int account_balance;

    public void showData() {
        //code to show data
    }

    public void deposit(int a) {
        account_balance = account_balance
    }
}
```

- Fields are private, it means they can only be accessed within the same class

Back to example 1

```
public class Account { private
   int account_number; private
   int account_balance;

   public void showData() {
      //code to show data
   }

   public void deposit(int a) {
   account_balance = account_balance + a
}
```

```
publ
   .
   A
   a
}
```

- Suppose a hacker managed to gain access code of your bank account and she tries to deposit amount -100.
- Is that possible?

```
publ
    .
    A
    a
}
```

Back to example 1

```
public class Account { private
   int account_number; private
   int account_balance;

   public void showData() {
      //code to show data
   }

   public void deposit(int a) {
       if (a < 0){
          //show error
       } else { account_balance =
       account_balance + a; }
}
```

Back to example 1

```
publi
    .
    Ac
    a
}
```

```
public class Account {
   private int account_number;
   private int account_balance;

   public void showData() {
      //code to show data
   }

   public void deposit(int a) {
       if (a < 0){
          //show error
       } else {
          account_balance = account_bala
       }
}
```

- The `deposit` method has a check for negative va

Back to example 1

```
public class Account { private
   int account_number; private
   int account_balance;

   public void showData() { //code
      to show data
   }

   public void deposit(int a) {
       if (a < 0){
          //show error
       } else { account_balance =
       account_balance + a; }
}
```

- Approach 1 and Approach 2 fail
- You never expose your data to an external party (wh
  secure) ▪ The entire code can be thought as capsu

```
public class Point{
  private int x;
  private int y;

  // constructor
  public Point(int initialX, int initialY){
    x = initialX; y = initialY;
  }

  // constructor
  public Point(){
  x = 0; y = 0;
  }

  // shifts points location by the given amount
  public void translate (int dx, int dy){ x +=
  dx; y += dy;
  }

  // computes the distance between two points
  public double distance(Point other){ int dx
  = x - other.x; int dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
} …
```

```
…
   /
  pu
  Po
  }

  pu

  }

  pu
  "
}
```

Poi

`private` Fields

- Declaring fields `private` encapsulates the state of
- `private` fields are visible to all the code inside the else

```
public class PointMain { public static void
     main(String[] args){
          //Create a Point objects
          Point p1 = new Point(5, 2);

          //Print each point
          System.out.println("p1.x is "+ p1.x)
     }

}
```

```
PointMain.java:6: error: x has private access
        System.out.println("p1.x is " + p1.x);
                                            ^
1 error
```

Accessing `private` fields

- Data members declared private can only be accesse
- No outside class can access them
- If you need to access these variables, you must use
- The “getter” are used to **retrieve** fields
- The ”setter” are used to **modify** fields

`get` and `set` for the `Account`

```
public class Account{ private int
        account_number; private int
        account_balance;

        // getter method public int
        getBalance() { return
        this.account_balance;
        }

        // setter method
        public void setNumber(int num)
           this.account_number = num;
        }
}
```

Accessing `private` fields

- We need to provide a way for the client code to acc

```
//A "read-only" access to the x
public int getX(){
          return x;
}
```

```
// Allows clients to change the
public void setX(int newX){ x =
}
```

- Client code will look more like this:

```
System.out.println(p1.getX());
p1.setX(14);
```

`get` and `set` methods

- Typically are used to retrieve or modify fields of a cla
- Not all fields need a get /set methods
- BUT, if you want to make sure that you restrict how your c you should think about using these methods

- Example `Point` class `int getX() {return x;} i`

  `setX(int xVal) {x = xVal;} void setY (int y`

Template of a well encapsulated d

```
public class <class name> {
   // fields
   private <type> <name>;
   private <type> <name>;
   ....
   //constructors
   public <class name>(<type> <name>, ...,<ty
          <statement>;
          .....
   }
   //methods
   public <type> <name>(<type> <name>,...,<ty
          <statement>;
          .....
   }
}
```

`Point` class

```
public class Point{
     private int x;
     private int y;

     public Point(){
          this(0, 0);
     }

     public Point(int x, int y){
          setLocation(x, y);
     }

     public double distanceFromOrigin(){
          return Math.sqrt(x * x + y * y);
     }

     public int getX(){
          return x;
```

```
… publ
     r
     }

     p

     }

     p

     }

     p
    dy

…
}
```

```
        }
...
```