Module 7: Objects

Ron K. Cytron

Department of Computer Science and Engineering
Washington University in Saint Louis
Thanks to Alan Waldman
for comments that improved these slides

Prepared for 2u
Semester Online

Copyright Ron K. Cytron 2013
7.0 Introduction

- So far, we have studied
  - data types, conditional and iterative execution
  - methods for computational abstraction (verbs)

- Relative low level
  - We meet the computer at its level
  - Why not get it to meet us at our level?

- What does this mean?
  - Make abstractions about nouns
    - robot.turn(90)
  - Create instances of related nouns
    - account1, account2, ...
  - Remember information between method calls
    - position of robot
    - balance in an account

- Objects allow us to do this
7.0 Introduction

• Java is an object-oriented language
  – But our introduction to this language was more concerned with building basic knowledge and skills
  – We now use what we know to design and instantiate objects

• We have already seen and used some objects:
  – Robot, Color
  – String (!)
    • Feels like a primitive type (int, double)
    • But each String is actually an object
7.0 Introduction

- **Example we will use**
  - **Account (as in bank account)**
    - Each account has a balance
    - Each account has its own interest rate

- **Why do I need objects? Can't I do this without them?**
  - **Yes!**
    - `int[] accountBalances (or int accountBalance1, ...)`
    - `double[] accountInterestRates (or double accountInterestRate1...)`

- **Problems?**
  - An account is the "same index" used in both of those arrays. It would be easy to make a mistake indexing the arrays. *We need a better organization that brings an account's balance and interest rate together in one object.*
  - The data in the arrays is open for change anywhere in the program. I want to control access so that an account balance changes by depositing money.
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}

• We see familiar elements
  • Declarations (names)
  • Methods
  • Braces for scope
7.1 Overview of an object

public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

• We see familiar elements
  • Declarations
  • Methods
  • Braces for scope
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
public class Account {

private int balance;

public Account(int initialBal) {
    this.balance = initialBal;
}

public int getBalance() {
    return this.balance;
}

public void deposit(int funds) {
    this.balance = this.balance + funds;
}
}

7.1 Overview of an object

• We see new elements
  • private
  • no return type
  • no use of static
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }
    public int getBalance() {
        return this.balance;
    }
    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

7.1 Overview of an object

- We see new elements
  - private
  - no return type
  - no use of static
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

7.1 Overview of an object

• We next look at these elements, one by one

• Keep in mind the purpose of a bank account
7.1 Overview of an object

```java
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }
}
```
public class Account {

• This specifies the name of the object type.
  • Style: Object type names always begin with an upper-case letter
  • No spaces or punctuation in the class name, but CamelCasing is used
    • Robot, Color, MouseListener
• The name must agree with the filename shown in the package explorer

}
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }

}
7.1 Overview of an object

```java
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```

- The constructor has the same name as the object type.
- It is like any other method
  - Can have parameters
  - But it has no return type
- It is responsible for giving birth to the object
  - Initializing instance variables
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }

}
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }

    }
public class Account {

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

    7.1 Overview of an object

    • The constructor has the same name as the object type.
    • It is like any other method
        • Can have parameters
        • But it has no return type
    • It is responsible for giving birth to the object
        • Initializing instance variables
}
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }

    // How do you find an object's constructors?
}

7.1 Overview of an object
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
public class Account {

    // Open an account with an initial balance

    public Account(int initialBal) {
    
    }

    // How do you find an object's constructors?
    
    • Look for a method with the same name as the object type
    • It has no return type
    • There can be more than one constructor!
7.1 Overview of an object

public class Account {

    private int balance;

    public Account() {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }

    // Open an account with zero balance

    public Account() {
    }

    // How do you find an object's constructors?
    // Look for a method with the same name as the object type
    // It has no return type
    // There can be more than one constructor!

}
public class Account {

    private int balance;

    public Account() {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

    // Open an account with zero balance

    public Account() {
    }

    // How do you find an object's constructors?
    
    • Look for a method with the same name as the object type
    • It has no return type
    • There can be more than one constructor!

}
7.1 Overview of an object

```java
public class Account {
    private int balance;

    public Account(Account closing) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```

// Open an account by transfer of funds

```java
public Account(Account closing) {
}
```

• How do you find an object's constructors?
  • Look for a method with the same name as the object type
  • It has no return type
  • There can be more than one constructor!

This signature differs from the other two as well
public class Account {


private int balance;

}
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
public class Account {

private int balance;

• Instance variables are usually declared private
  • So that they can be accessed only within a method
• They live "between the methods"
  • Methods can read and write them
  • Their values persist after methods return
• They are the "has-a"s of object oriented design
  (more on this later…)

}
public class Account {
    // An Account has-a balance
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }
}

7.1 Overview of an object

- Instance variables are usually declared private
  - So that they can be accessed only within a method
- They live "between the methods"
  - Methods can read and write them
  - Their values persist after methods return
- They are the "has-a"s of object oriented design (more on this later...)
7.1 Overview of an object

```java
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }
}
```
public class Account {

    private int balance;

    public Account(int initialBal) {

    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

7.1 Overview of an object

- Constructor is obligated to initialize the instance variables
public class Account {

private int balance;

public Account(int initialBal) {
    this.balance = initialBal;
}

public int getBalance() {
    return this.balance;
}

public void deposit(int funds) {
    this.balance = this.balance + funds;
}
}

7.1 Overview of an object

This assignment statement initializes the instance variable balance

• The value for initialization is provided as a parameter to the constructor
• new Account(100) causes this constructor to initialize the account balance to 100

Recall
• Constructor is obligated to initialize the instance variables
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

• This assignment statement initializes the instance variable balance
  • The value for initialization is provided as a parameter to the constructor
  • new Account(100) causes this constructor to initialize the account balance to 100
7.1 Overview of an object

```java
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```

- This assignment statement initializes the instance variable balance
  - The value for initialization is provided as a parameter to the constructor
  - `new Account(100)` causes this constructor to initialize the account balance to 100

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

This assignment statement initializes the instance variable balance
- The value for initialization is provided as a parameter to the constructor
- new Account(100) causes this constructor to initialize the account balance to 100
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }

    new Account(100)
    The instance variable balance now has value 100, reflecting the creation of this new Account

    • This assignment statement initializes the instance variable balance
      • The value for initialization is provided as a parameter to the constructor
      • new Account(100) causes this constructor to initialize the account balance to 100

}
public class Account {

private int balance;  \(100\)

public Account(int initialBal) {
    this.balance = initialBal;
}

• Instance variables retain their values
  • So when the constructor returns, the balance is still 100
  • The balance value persists and any method in Account can access or change it
7.1 Overview of an object

```java
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
```

- References to instance variables
  - Can occur in any (non-static) method
  - Use the current value of the instance variable
public class Account {

private int balance;

public Account(int initialBal) {
    this.balance = initialBal;
}

public int getBalance() {
    return this.balance;
}

public void deposit(int funds) {
    this.balance = this.balance + funds;
}

}

7.1 Overview of an object

- References to instance variables
  - Can occur in any (non-static) method
  - Use the current value of the instance variable

• References to instance variables

• Can occur in any (non-static) method

• Use the current value of the instance variable
7.1 Overview of an object

```java
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```

- References to instance variables
  - Can occur in any (non-static) method
  - Use the current value of the instance variable

7.1 Overview of an object • References to instance variables
  • Can occur in any (non-static) method
  • Use the current value of the instance variable
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

• How do you find the instance variables of an object?
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

    }

7.1 Overview of an object

• Instance variables
  • Are declared outside of any method
  • Usually have the word private in their declaration
  • Usually start with "this."
7.1 Overview of an object

```java
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance += funds;
    }
}
```

• We write methods to provide functionality for the object
• These methods can use the instance variables as needed, or change their value
  • Static methods cannot access instance variables
• They have signatures, return types, and code just like the methods we have studied so far
7.1 Overview of an object

```java
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
```

- This method returns the value of the instance variable balance
- When method serves only to return the value of an instance variable, we call it an accessor, or more specifically, a getter
7.1 Overview of an object

public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

A call to getBalance() returns the current value of the instance variable balance.
public class Account {

private int balance;

public Account(int initialBal) {
    this.balance = initialBal;
}

public int getBalance() {
    return this.balance;
}

public void deposit(int funds) {
    this.balance = this.balance + funds;
}

}
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
7.1 Overview of an object

```java
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```

A call to `getBalance()` returns the current value of the instance variable `balance` (say, 100)
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

7.1 Overview of an object

- This method serves to deposit money into an account.
- It does this by changing the value of the balance instance variable.
- The code increments this object's balance by the specified value of funds.
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

7.1 Overview of an object

Suppose balance is currently 100, and we ask this Account to deposit 50
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}

7.1 Overview of an object

Suppose balance is currently 100, and we ask this Account to deposit 50
7.1 Overview of an object

```java
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```

Suppose `balance` is currently 100, and we ask this `Account` to deposit 50.
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

    }
7.1 Overview of an object

public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
7.1 Overview of an object

Suppose balance is currently 100, and we ask this Account to deposit 50
Now balance is 150
7.1 Overview of an object

```java
public class Account {
    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```

The value persists after the method returns.
7.1 Overview of an object

```java
public class Account {
    private int balance;
    public Account(int initialBal) {
        this.balance = initialBal;
    }
    public int getBalance() {
        return this.balance;
    }
    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```
7.1 Overview of an object

```java
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}

Instance variables
```
7.1 Overview of an object

```java
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }
}
```

Constructors
public class Account {

    private int balance;

    public Account(int initialBal) {
        this.balance = initialBal;
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int funds) {
        this.balance = this.balance + funds;
    }

}
7.2 Exercise

• Video intro. All following are for question card
• Showing the previous code, I'd like to have students identify
  – The constructor
  – The instance variable(s)
  – An accessor
  – A method that changes the value of an instance variable

• Suppose the balance of an account represents dollars, and it costs one dollar to find out your balance.
  – Modify the getBalance() method appropriately
7.2 Exercise

• Video response
  – Show solution
7.3 How to make and use objects

Account acct1 = new Account(100);

• Declarations are just like type types we have already studied:
  
  type name = initial value

• This is just like declarations we have seen before……
7.3 How to make and use objects

```c
int size = 300;
```

- This is an `int` variable named `size`
- It can take on any `int`'s value
- It happens to be initialized to 300
- But it could change to some other value later

- Declarations are just like type types we have already studied:
  ```c
type name = initial value
  ```
7.3 How to make and use objects

Account acct1 = new Account(100);

• This is an Account variable named acct1
• It can take on any Account's value
• It happens to be initialized so that it references new Account(100)
• But it could reference some other Account later

• Declarations are just like type types we have already studied:
  type name = initial value
• Let's break this down step by step
7.3 How to make and use objects

Account acct1 = new Account(100);

- Declarations are just like type types we have already studied:
  
  type name = initial value

- By saying `new Account(...)` we run the `constructor` for the `Account` object
7.3 How to make and use objects

Account acct1 = new Account(100);

• Java looks for a constructor with the signature of an int parameter. Recall:

```java
public Account(int initialBalance) {
    this.balance = initialBalance;
}
```

• Declarations are just like type types we have already studied:

```java
type name = initial value
```

• By saying `new Account(...)` we run the constructor for the Account object
7.3 How to make and use objects

Account acct1 = new Account(100);

• Java looks for a constructor with the signature of an int parameter. Recall:

```java
public Account(int initialBalance) {
    this.balance = 100;
}
```

• Declarations are just like type types we have already studied:

```java
    type name = initial value
```

• By saying new Account(...) we run the constructor for the Account object
7.3 How to make and use objects

Account acct1 = new Account(100);

• The result of this operation is a new Account object whose balance instance variable is initialized to 100

• Declarations are just like type types we have already studied:
  
  type name = initial value

• By saying new Account(...) we run the constructor for the Account object
7.3 How to make and use objects

Account acct1 = new Account(100);

- By saying `new Account(...)`, we run the constructor for the Account object.
- The result of the constructor's call is returned.
- That value is captured in `acct1`, which now references the constructed object.
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

• Now there are two objects
  • acct1 has an initial balance of 100
  • acct2 has an initial balance of 50
• These are separate objects and they cannot change or reference each other
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

```java
public Account(int initialBalance) {
    this.balance = initialBalance;
}
```

- Common analogy:
  - Constructor is a cookie cutter
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

public Account(int initialBalance) {
    this.balance = initialBalance;
}

• Common analogy:
  • Constructor is a cookie cutter
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

• Common analogy:
  • Constructor is a cookie cutter
  • The constructed objects are cookies
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

• Common analogy:
  • Constructor is a cookie cutter
  • The constructed objects are cookies
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

• Common analogy:
  • Constructor is a cookie cutter
  • The constructed objects are cookies
Account acct1 = new Account(100);
Account acct2 = new Account(50);

• We will next run a method on the object referenced by acct1.
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);

• Runs the deposit method on the object referenced by acct1.
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);
acct1.deposit(35);
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);
acct1.deposit(35);

- **Syntax**
  - Mention an object reference, followed by a dot
  - Then mention a method defined for that object, with the parameters it needs
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);

Recall:
public void deposit(int funds) {
    this.balance = this.balance + funds;
}

Throughout, this is referring to acct1
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);

Recall:

```java
public void deposit(int funds) {
    this.balance = this.balance + funds;
}
```
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);

Recall:

```
public void deposit(int funds) {
    this.balance = 100 + funds;
}
```
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);

Recall:

public void deposit(int funds) {
    this.balance = 100 + funds;
}

100
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);

Recall:

```
public void deposit(int funds) {
    this.balance = 100 + 35;
}
```
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);

Recall:

```java
public void deposit(int funds) {
    this.balance = 135;
}
```
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);

Recall:

```java
public void deposit(int funds) {
    this.balance = 135;
}
```
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);
acct1.deposit(35);

• Runs the deposit method on the object referenced by acct1.
• Result is that the balance increases to 135
• The method is void, so it returns nothing
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);
System.out.println("Balance 1 " + acct1.getBalance());

• Running getBalance() on acct1 returns that account's balance, as held in its balance instance variable
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);
System.out.println("Balance 1 " + acct1.getBalance());

• Causes the following message to print:
  
  Balance 1 135
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);
System.out.println("Balance 1 " + acct1.getBalance());

Account acct3 = new Account(acct2.getBalance()*3);

• Evaluated first, returns 50
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);
System.out.println("Balance 1 " + acct1.getBalance());

Account acct3 = new Account(acct2.getBalance() * 3);

• Evaluated first, returns 50
• 50 * 3 = 150
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);
System.out.println("Balance 1 " + acct1.getBalance());

Account acct3 = \textbf{new Account(acct2.getBalance()*3)};

\begin{itemize}
\item Evaluated first, returns 50
\item 50 * 3 = 150
\item Constructor is run on 150
\end{itemize}
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);
acct1.deposit(35);
System.out.println("Balance 1 " + acct1.getBalance());

Account acct3 = new Account(acct2.getBalance() * 3);

- Evaluated first, returns 50
- 50 * 3 = 150
- Constructor is run on 150 (another cookie cut)
- Result is now referenced by acct3
7.3 How to make and use objects

Account acct1 = new Account(100);
Account acct2 = new Account(50);

acct1.deposit(35);
System.out.println("Balance 1 "+ acct1.getBalance());

Account acct3 = new Account(acct2.getBalance()*3);
System.out.println("Balance 3 "+ acct3.getBalance());

• Prints the following
  Balance 3 150
7.3b Roundtable

• I tell a story, and the user instantiates objects on the screen
7.4 How are objects stored?

- **Stack (review)**
  - Like a stack of plates in the cafeteria
  - Parameters are pushed onto the stack for a method call
    - `foo(5,7)`: 5 is pushed and then 7
  - The method pops the values off the stack and assigns them to its parameters
    - For `foo(a,b)`, `a` is assigned 5, `b` is assigned 7
  - The method executes up to a `return` statement
  - The value returned by the method is pushed onto the stack
  - The caller retrieves the value
    - `x = foo(a,b)`: the value computed by `foo` is popped from the stack and assigned to `x`
7.4 How are objects stored?

- **Stack (review)**
  - Like a stack of plates in the cafeteria

- **Heap (new)**
  - Objects are allocated in the heap
  - Objects live there until they can no longer be referenced
    - After they die, they are considered *garbage* and are collected by the *garbage collector*
  - The space allocated for an object includes room for its instance variables
7.4 How are objects stored?

```java
Account acct1 = new Account(100);
```

Heap

```
acct1
```

```
100
```
7.4 How are objects stored?

Account: `acct1 = new Account(100);`

Heap: `acct1`
7.4 How are objects stored?

Account acct1 = new Account(100);

- The local variable acct1 now references the Account object allocated in the heap
- Another way to think of this is as follows.....
7.4 How are objects stored?

```java
Account acct1 = new Account(100);
```

- Every object in the heap is situated at some address
- In this case, the Account object is allocated at address **348**
7.4 How are objects stored?

```java
Account acct1 = new Account(100);
```

- Every object in the heap is situated at some address
- In this case, the Account object is allocated at address **348**
- The `acct1` variable therefore contains **348**
7.4 How are objects stored?

Account acct1 = new Account(100);
Account acct2 = new Account(50);

- Space is made in the heap for the second allocation
7.4 How are objects stored?

Account acct1 = new Account(100);
Account acct2 = new Account(50);
7.4 How are objects stored?

```java
Account acct1 = new Account(100);
Account acct2 = new Account(50);
```

Diagram:
- acct1
  - Value: 348
- acct2
  - Value: 408
- Heap
  - Value: 100
  - Value: 50
7.4 How are objects stored?

• Each object allocated in the heap has room for all of its instance variables

```java
public class Account {
    private int balance;
}
```
7.4 How are objects stored?

- Each object allocated in the heap has room for all of its instance variables
- The more instance variables, the more space is needed for each object

```java
public class Account {
    private int balance;
    private double interestRate;
}
```
7.4 How are objects stored?

- The constructor should be modified to take in an interest rate and store it as the instance variable

```java
public class Account {
    public Account(int iBal, double r) {
        this.balance = iBal;
        this.interestRate = r;
    }
}
```
7.4 How are objects stored?

- The constructor should be modified to take in an interest rate and store it as the instance variable.
- Or, perhaps the interest rate could be computed from the initial balance?

```java
public class Account {
    public Account(int iBal) {
        this.balance = iBal;
        this.interestRate = iBal/1000.0;
    }
}
```
7.4 How are objects stored?

These are issues of design
Satisfying the method's specification

```java
public class Account {
    public Account(int iBal) {
        this.balance = iBal;
        this.interestRate = iBal/1000.0;
    }
}
```
7.4b Exercises with object placement

• Go through about 10 lines of code
  – Drawing pointers
  – Making up addresses for things
7.5 Object design

- We have declared and used Account objects.

- How are such objects designed?
  - User story: describes the object in terms of its properties and the functionality it offers.
  - We respond to the user story by defining:
    - Instance variables, hidden from the outside world, that allow our object to do its business.
    - Methods, visible to the outside world, that allow the outside world to interact with our object.

- In design, there are no right or wrong solutions.
  - But each solution usually offers its own advantages and disadvantages.
7.5 Object design

• An Account has a balance
  – "has-a" is code for "needs an instance variable"
  – The instance variable holds the property
  – It is usually initialized in the constructor
    • Based on the user story

• An Account has an initial balance
  – When any Account object is constructed, it must have some initial balance
7.5 Object design

• I need an Account object

```java
public class Account {
}
```
7.5 Object design

• I need an Account object
• An account has-a balance

```java
public class Account {
    private int balance;
}
```
7.5 Object design

- I need an Account object
- An account has-a balance

```java
public class Account {
    private int balance;
}
```

- What type should we use for this instance variable?
- The story isn't clear
- So I picked int, but we could change this later if necessary
7.5 Object design

- I need an Account object
- An account has-a balance
- Accounts are created with a specified initial balance

```java
public class Account {

    private int balance;

    public Account(int b) {
        this.balance = b;
    }
}
```
7.5 Object design

• I need an Account object
• An account has-a balance
• Accounts are created with a specified initial balance
• Or, not! Assume 0

```java
public class Account {
    private int balance;

    public Account(int b) {
        this.balance = b;
    }

    public Account() {
        this.balance = 0;
    }
}
```
7.5 Object design

- I need an Account object
- An account has-a balance
- Accounts are created with a specified initial balance
- Or, not! Assume 0

```java
public class Account {

    private int balance;

    public Account(int b) {
        this.balance = b;
    }

    public Account() {
        this(0);
    }
}
```

A constructor can defer to another constructor using `this(.....);`
7.5 Object design

- I need an Account object
- An account has-a balance
- Accounts are created with a specified initial balance
- Or, not! Assume 0
- You can find out the balance of an account

```java
public class Account {

    private int balance;

    public Account(int b) {
        this.balance = b;
    }

    public Account() {
        this(0);
    }

    public int getBalance() {
        return this.balance;
    }
}
```
7.5 Object design

• I need an Account object
• An account has-a balance
• Accounts are created with a specified initial balance
• Or, not! Assume 0
• You can find out the balance of an account
• You can deposit funds

public class Account {

    private int balance;

    public Account(int b) {
        this.balance = b;
    }

    public Account() {
        this(0);
    }

    public int getBalance() {
        return this.balance;
    }

    public void deposit(int f) {
        this.balance += f;
    }
}

126
7.5 Object design

- Add instance variables as needed (has-a)
  
  ```java
  private int x, y;
  ```
Add instance variables as needed (has-a)

```java
private int x, y;
double a, b, c;
```
7.5 Object design

- Add instance variables as needed (has-a)
- Add methods to provide desired functionality
7.5 Object design

- Add instance variables as needed (has-a)
- Add methods to provide desired functionality

Account

getBalance()
deposit(int funds)
boolean withdraw(int c)
7.5 Object design

- Add instance variables as needed (has-a)
- Add methods to provide desired functionality

```java
Account
  transfer(Account a, int funds)
  getBalance()
  deposit(int funds)
  boolean withdraw(int c)
```

End of Oyster
7.6 Exercise

• Video intro
• Question card
  – Come up with an object of interest similar to Account
• Think about
  – Its instance variables
  – The functionality it offers
7.6 Exercise

• Video response
  – None – we will see suitable responses in the roundtable that comes next
  – Roundtable with a student discussing the design
7.6b Explanation of Object Design

- Take one object design and go over it with a student, interactively, exploring other design choices


7.7 Scope and instance variables

- Go over scope with a student
  - Instance variables
  - Local variables

- When are we talking about which?
  - Using `this` to mean instance variables
  - Using eclipse to find the one we mean

Roundtable 7

End of Roundtable
7.8 Conclusion

• Object
  – Collect related behaviors
  – Model nouns, concepts of interest

• Components
  – Instance variables
    • Typically private, accessed only within a class
  – Methods
    • Typically public, offer functionality to outside world

• Syntax for object usage
  – object.doSomething(parameters)

• As an exercise, regard the world in terms of objects
  – door.open()
  – person.sendText("hey")