Motivation

• Suppose the states of a bank account is represented by some variables:
  ```
  // bank account
  int balance;  // current balance
  int deposits; // deposits to date
  int withdrawals; // wd to date
  ```

• Shortcomings:
  • Relationships among variables are not made explicit
  • Scale: if we want 2 bank accounts, then we write twice as much code:
  ```
  // bank account 1
  int balance1;
  int deposits1;
  int withdrawals1;
  // bank account 2
  int balance 2;
  int deposits2;
  int withdrawals2;
  ```

Aggregation

• Group variables into a new abstraction that makes their relationship to one another explicit
  • Such an abstraction is called a class
  • Abstraction: a named compound thing that can be manipulated as a unit
  • Example: a class Account

• Scale: create multiple instances of the abstraction
  • Each instance of the abstraction is called an object
  • Example: refer to the two accounts as account1 and account2

Terminology and concepts

• Object: contains variables (fields, instance variables) and methods
  • Variables: “state” or “characteristics”
  • Methods: “behavior” or “action”
  • e.g., name, age

• Class: blueprint (definition) of an object
  • No memory space is reserved for object data

• Imagine a class Cookie. To make a whole lot of cookies, you may want to
  • Make a cookie cutter—define the class
  • Need to actually stamp out the cookie— instantiate an object
  • Note that class definition ≠ object instantiation
Variables

Two main types of variables:
- Primitive type
- Reference to object

Some variables with different properties:
- Local: live and die inside a method
- Instance variable: owned by and accessed through individual instances (objects)
- Static variable: class variable shared by all instances—only one copy in a class

Class Definition

```
public class class-name {
  declaration (and initialization)
  constructor
  methods
}
```

Variables and values revisited

- **Variable**: named place to hold a value
- Name: `value`
- Values of primitive types are held directly in variables
  - `count 0`: an int variable
- Values of non-primitive types are references to objects shown graphically by arrows
  - `account1` an Account variable
  - `balance 0`
  - `deposits 0`
  - `withdrawals 0`

Class definitions: declarations

```
public class Account {
  private int balance; // current balance
  private int deposits; // deposits to date
  private int withdrawals; // withdrawal to date
}
```

- Declarations of a class define **fields** (instance variables) of the class
- Each class is a type. Classes are not primitive types.

Declarations Revisited

- **Syntax**: `type name;`
- Examples:
  - `int count;`
  - `Account account1; Account account2;`
- Instance variables have default initial values
  - `int variables: 0`
- Non-primitive (reference) variables: `null`
- Value `null` signifies that no object is referenced
Declaration and object instantiation (initialization)

- Syntax: \( \text{type name} = \text{expression} \);
- Examples:
  ```java
  int count = 0;
  Account account1 = new Account();
  Account account2 = new Account();
  ```

Object instantiation:
- An expression of the form \( \text{new class-name()} \) computes a reference to a newly created object of the given class.

Manipulating a field of an object

- Let \( f \) be a field defined for class \( c \)
- Let \( x \) refer to an object \( o \) of class \( c \)
- Then \( x.f \) is a variable of object \( o \)
- The dot (.) means “follow the arrow”
- Example:
  ```java
  // deposit d into account1
  account1.balance = account1.balance + d;
  account1.deposits = account1.deposits + d;
  ```
  ```java
  // shortcut
  account1.balance += d;
  account1.deposits += d;
  ```

References are values

- Suppose you declare \( a \) to be an Account reference variable:
  ```java
  Account a;
  ```
- Then you can assign an reference to an Account object into variable \( a \)
- Example:
  ```java
  // if k is 1, deposit d into account1
  // otherwise deposit d into account2
  if (k==1)
    a = account1;
  else
    a = account2;
  ```
  ```java
  // deposit d to Account a
  a.balance = a.balance + d;
  a.deposits = a.deposits + d;
  ```

References are values, cont’d

```
| k | 2 | an int variable |
| a | null | an Account variable |
| account1 | an Account variable |
| balance | 0 |
| deposits | 0 |
| withdrawals | 0 |
```

```
| account2 | an Account variable |
| balance | 0 |
| deposits | 0 |
| withdrawals | 0 |
```
Methods

A method is a named, parameterized group of statements.

Syntax

```
return-type method-name ( parameter-list ) {  
    statement-list  
}  
```

`return-type void` means nothing is returned from the method.

Example class definition

```java
public class Account {
    private int balance;  // current bal.
    private int deposits; // deposits to date
    private int withdrawals; // withdrawal to date

    // deposit d to account
    public void deposit(int d) {
        balance = balance + d;
        deposits = deposits + d;
    }

    // withdraw w from account
    public void withdraw(int w) {
        balance -= w;
        withdrawals += w;
    }
}
```