Computer Science 1

Lecture 7

Objects


KEEP
CALM
AND TRUST
THE
RECURSION

Recursion idea

```
public static <type> f(x) {
    if (base_case(x))
        return result;
    else {
        y = reduce(x);
        return … f(y) …;
    }
}
```

Is this input for which the answer is trivial?

Brings the input closer to the trivial case.

Compute the answer for the original input using the answer for the simpler case.

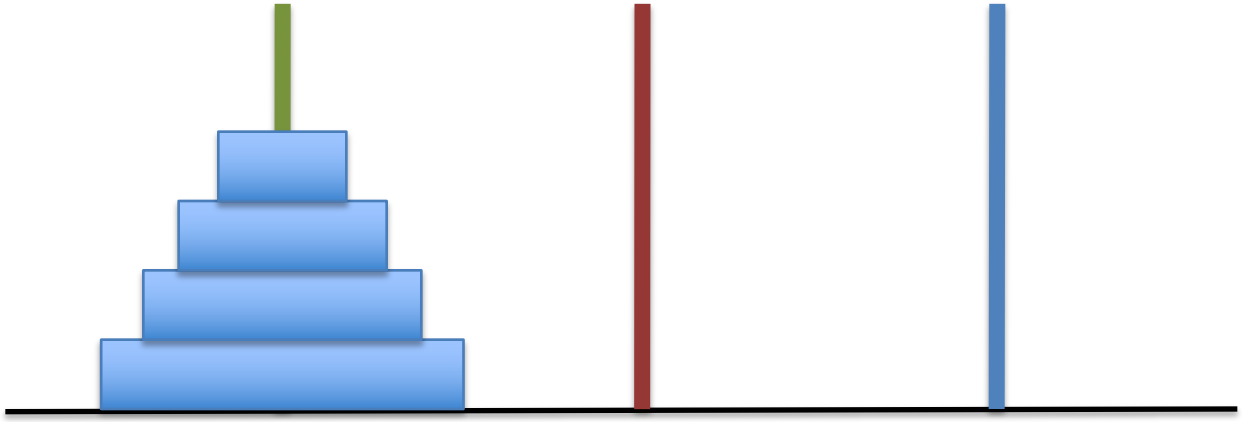
If you haven't done so yet, read the recursion chapter from the book!

Recursive Solution

```
public static int recursiveSeries(int x) {
    System.out.print(x + " ");

    if (x == 1)
        return 0;
    else
        if (x%2 == 0)
            return recursiveSeries(x/2) + 1;
        else
            return recursiveSeries(3*x+1) + 1;
}
```

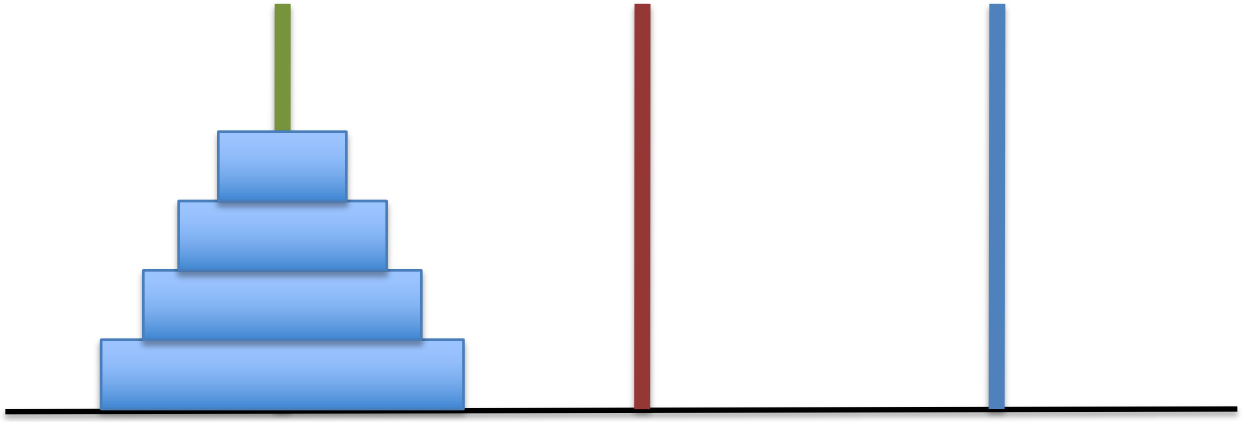
```
int number = ...;
int result = recursiveSeries(number);
System.out.println();
System.out.println("This took " + result + " steps.");
```

Towers of Hanoi



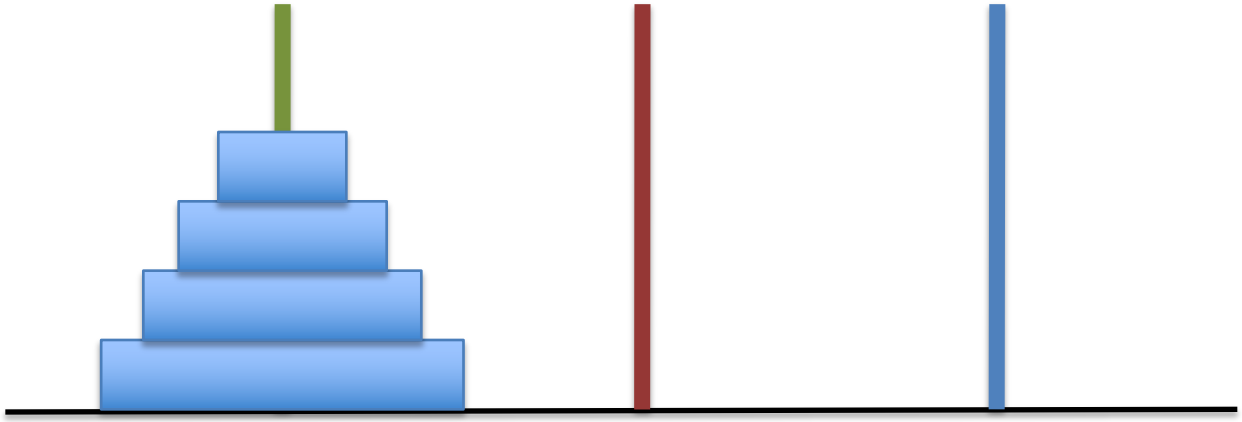
Towers of Hanoi: Principle

Move n discs from | to | using |



Towers of Hanoi: Principle (2)

Move n discs from | to | using |



1. Move n-1 discs from | to | using |
2. Move disc from | to |
3. Move n-1 discs from | to | using |

Overview

Objects & classes

- need
- definition
- declaration & construction
- use

Object Oriented Programming

Tools and TOOLS

In general

Programs deal with many similar “things”

- Multi-media: songs, movies, …
- Bank: clients, accounts, loans, …
- Communication: emails, contacts, folders

All store same type of data and exhibit similar behavior

- Song: length, bit-rate, can be played, info added, …
- Emails: sender, addressee, subject, can be send, forwarded, replied to, …

Guarding complexity

1. Define behavior only once

2. Hide complexity from rest of program
3. Protect integrity of data by restricting access

= Encapsulation

Object – Oriented Programming

Things = Objects

= Entities that are manipulated in programs

- like basic type values
- are referenced by variables, used as parameters, …

(can) Have an internal state

- variables with values connected to the object

(can) Have a set of abilities

- methods that can be called on the object

What kinds of things?

Can be data related:

- Songs, Movies, Pictures, … in a multi-media application
- Employees, Branches, Holdings, … in business administration

Can be behavior related:

- Filters (blur, sharpen, edge detection …) in image processing
- OpponentStrategies in computer games

Usually, it's both.

Object example: Strings

Each string is an *object*; it is an *instance* of the *class* String

- internal state = values of all characters
- abilities:

  Given a variable of type String called “str”

  – str.length()

  – str.toUpperCase()

  – str.compareTo(other) (where other is also String)

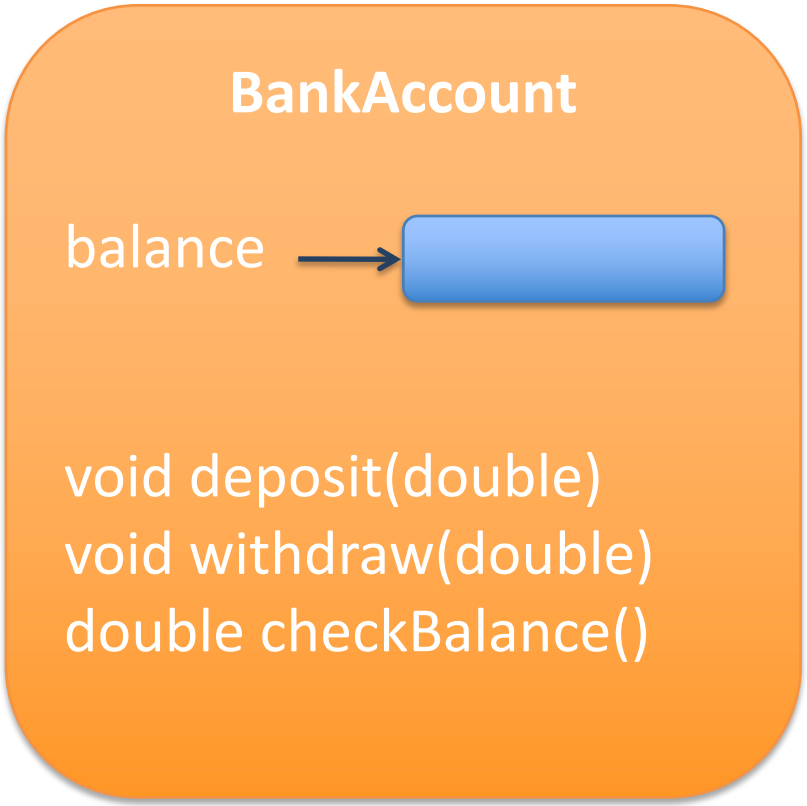
An example: BankAccount

Internal state

- balance

Abilities

- deposit
- withdraw
- check balance

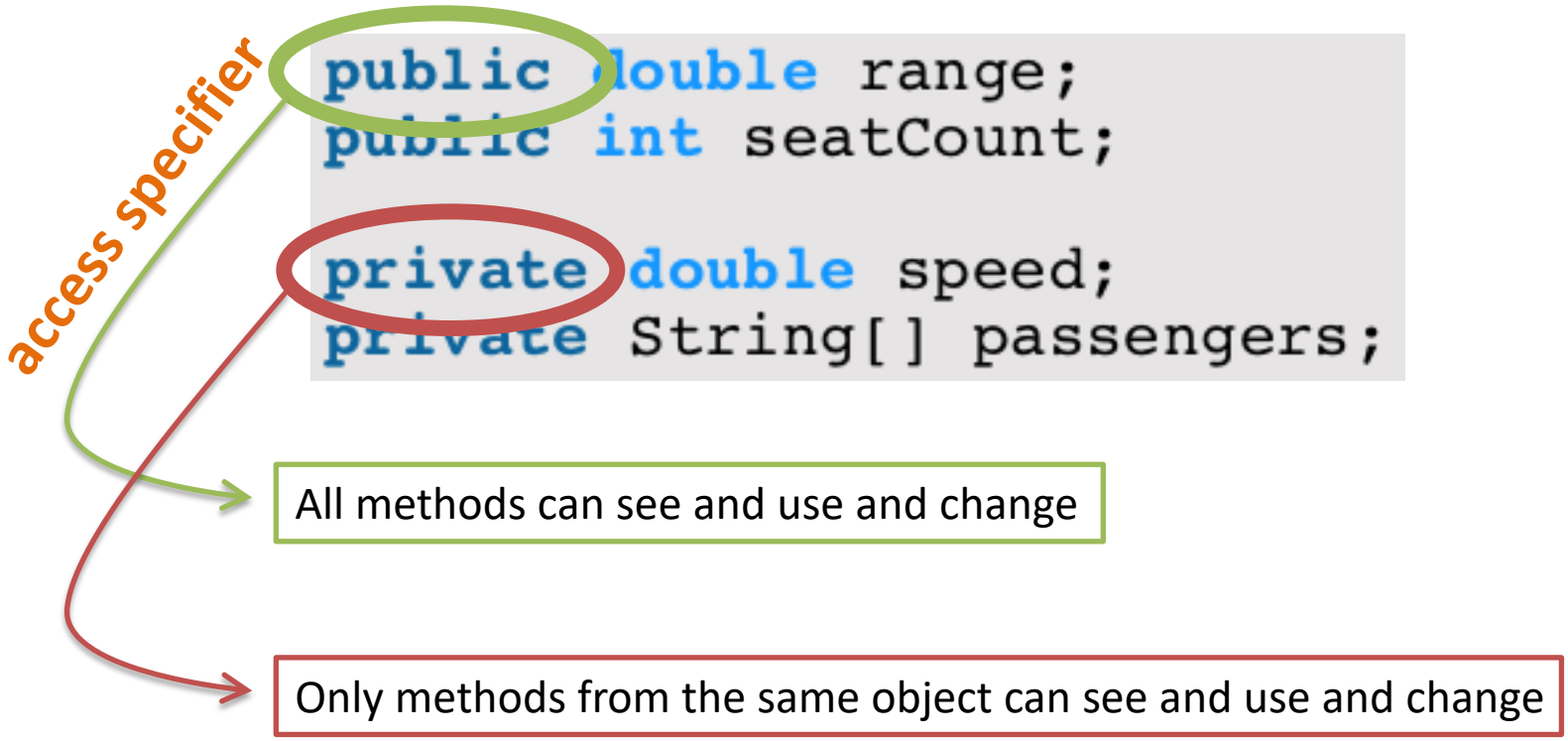


```java
public class BankAccount {

    private double balance;

    public BankAccount(double amount) {
        balance = amount;
    }

    public void deposite(double amount) {
        balance += amount;
    }

    public boolean withdraw(double amount) {
        if (amount > balance) {
            return false;
        } else {
            balance -= amount;
            return true;
        }
    }

    public double checkBalance() {
        return balance;
    }

}
```

Instance variables

= data stored in the objects

– lifetime is the same as the object!!!



Data encapsulation

Hiding/protecting data inside an object

```
Plane memphisBelle = new Plane();

System.out.println(memphisBelle.range);
memphisBelle.seatCount = 80;

System.out.println(memphisBelle.speed);
memphisBelle.passengers[0] = "Kurt";
```

- Safety
- Hides complexity
- Less bugs in code

In practice: make ALL instance variables **private**

Difference to local variables

Local and parameter variables belong to a method

- When the method runs, its local and parameter variables come to life
- When the method exits, they are removed immediately

Instance variables belong to an object, not methods

- When an object is constructed, its instance variables are created
- The instance variables stay alive until no method uses the object any longer

Instance variables are initialized to a default value, but you must initialize local variables

Instance methods

Implement the objects abilities

– are invoked on an instance of the class

```
public void deposite(double amount) {
    balance += amount;
}
```

```
public static void main(String[] args) {

    BankAccount myAccount = new BankAccount(100.0);

    myAccount.deposit(50.0);

    ...
}
```

Getters and Setters

Are used to provide access to private instance variables

```
private double balance;

public double getBalance() {
    return balance;
}

public void setBalance(double in) {
    balance = in;
}
```

Makes life easier?

Can hide implementation!

```
public class Angle {

    private double angle;

    public void setInRadians(double in) {
        angle = ... in ...;
    }
    public double getInRadians() {
        return ... angle ...;
    }
    public void setInDegrees(double in) {
        angle = ... in ...;
    }
    public double getInDegrees() {
        return ... angle ...;
    }
}
```

Implicit parameter

The object on which the method is invoked

```
public void deposite(double amount) {
    balance += amount;
}
```

```
public static void main(String[] args) {

    BankAccount myAccount = new BankAccount(100.0);

    myAccount.deposit(50.0);

    ...
}
```

implicit

explicit

this

Reserved word that gives an explicit reference to the implicit parameter

```
balance += amount;
```

```
this.balance += amount;
```

- can make code easier to read
- instance methods invoked without implicit parameter inside another instance method of the same class use *this* as implicit parameter

Constructor

special method: used to make a new instance of a class – possibility for some initialization!

```
public BankAccount(double amount) {
    balance = amount;
}
```

- has no return type; returns an instance of class
- name = name of class
- can be overloaded! Different constructors all use same name but can have different parameters.

Default constructor

If no constructor is declared, Java provides a default one

- no parameters
- all instance variable get default values
  - 0 or 0.0 for numeric variables
  - null for reference variables (objects, including arrays)

ONLY WHEN NO OTHER CONSTRUCTOR IS DECLARED!!

When you need 2 constructors, one without parameters and one with, you have to declare **both** yourself!

more this

- can also be used to invoke a constructor from another method (or constructor)
  - pretty advanced Java; will make sense at some point in your programming career

- can be used to reference “shadowed” instance variables

```
public BankAccount(double balance) {
    this.balance = balance;
}
```

Class Declaration

```
<access specifier> class <className> {

    <instance variables>

    <constructors>

    <methods>

}
```

Example: Counter

Figure 1 A Tally Counter

```
public class Counter {

   private int value;

   public Counter() {
      value = 0;
   }

   public Counter(int initialValue) {
      value = initialValue;
   }

   public void count() {value++;}

   public int getValue() {return value;}

}
```

implementation = private

interface = public

Commenting the public interface

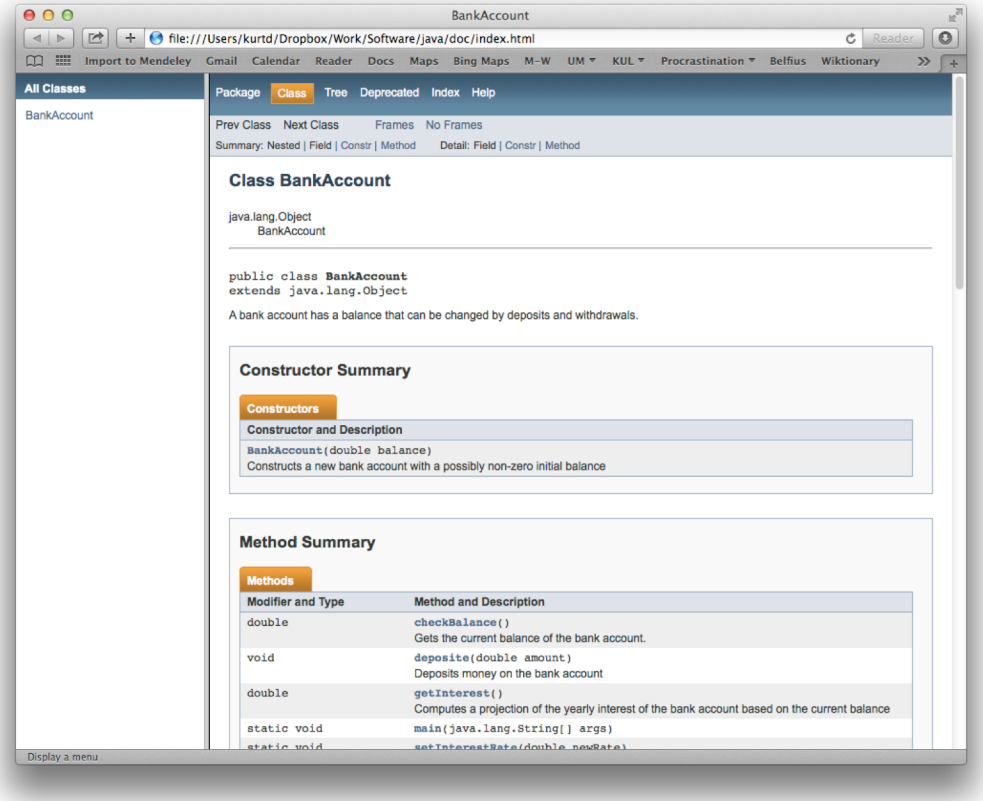
```
...
/**
    Withdraws money from the bank account if sufficient
    funds are available.  Else nothing happens.
    @param amount the amount to withdraw
    @return whether the withdrawal was succesful
*/
public boolean withdraw(double amount) {
    if (amount > this.balance) {
        return false;
    } else {
        this.balance -= amount;
        return true;
    }
}
/**
    Gets the current balance of the bank account.
    @return the current balance
 */
public double checkBalance() {
    return this.balance;
}
...
```

Javadoc & API

javadoc automatically creates API like html-pages for your classes

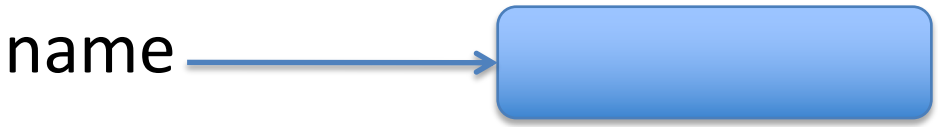
Provide comments for

- every public class
- every public method
- every parameter
- every return value

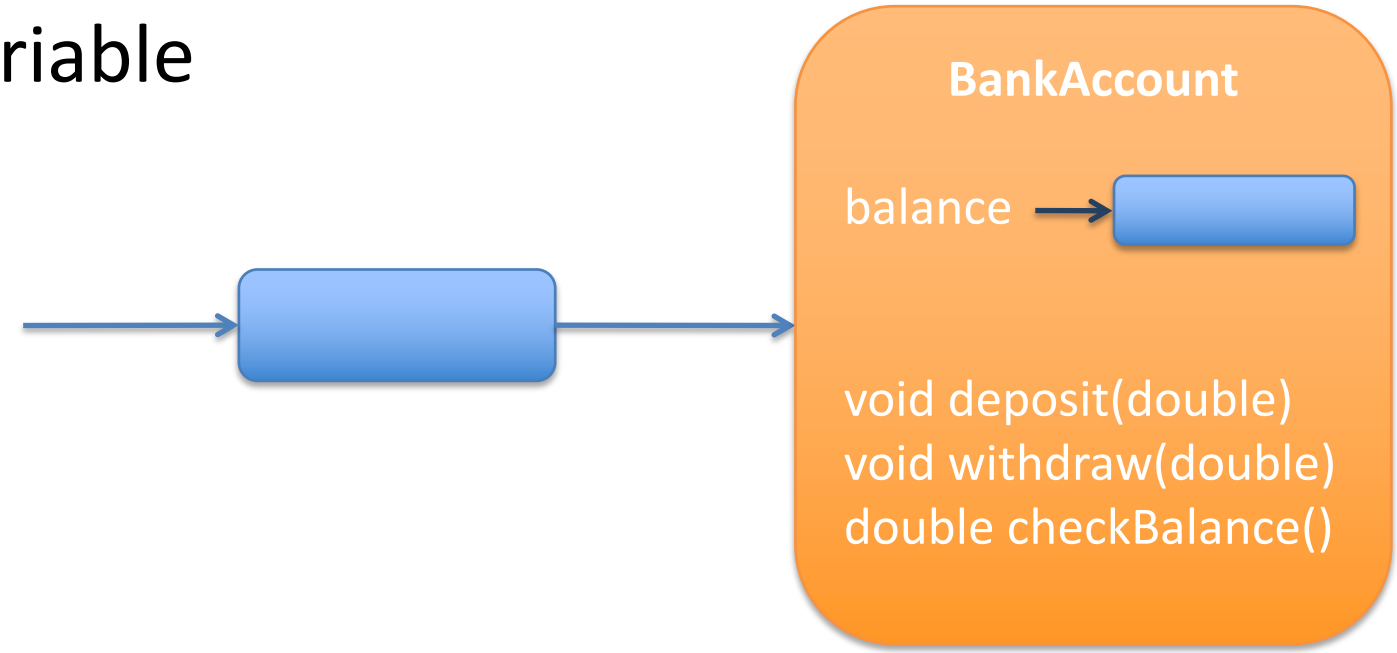


Object variables vs. basic variables

Basic variable



Object variable

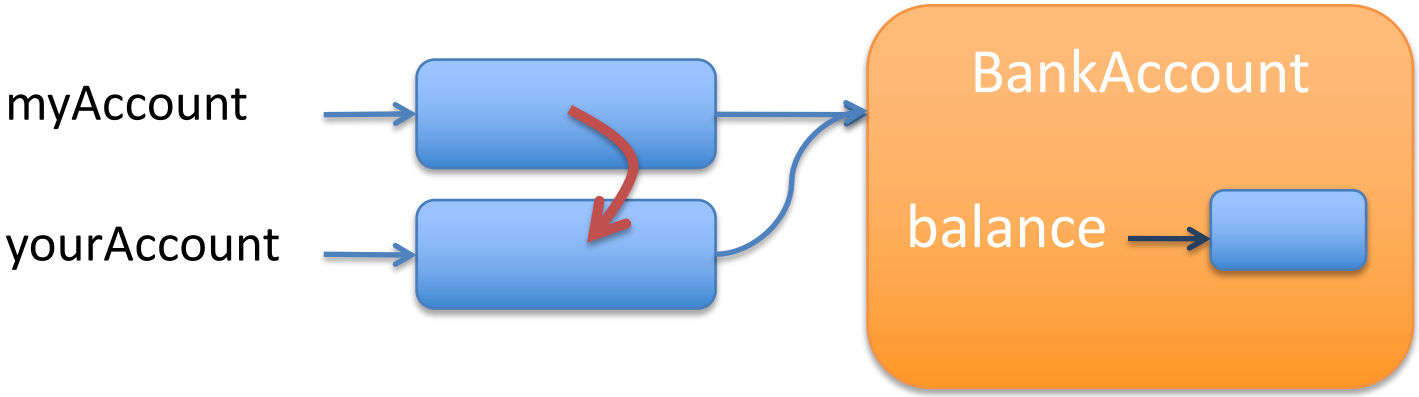


Objects references

```
BankAccount myAccount = new BankAccount(100.0);
BankAccount yourAccount = myAccount;

System.out.println(yourAccount.getBalance());

yourAccount.withdraw(50.0);
System.out.println(myAccount.getBalance());
```

100

50



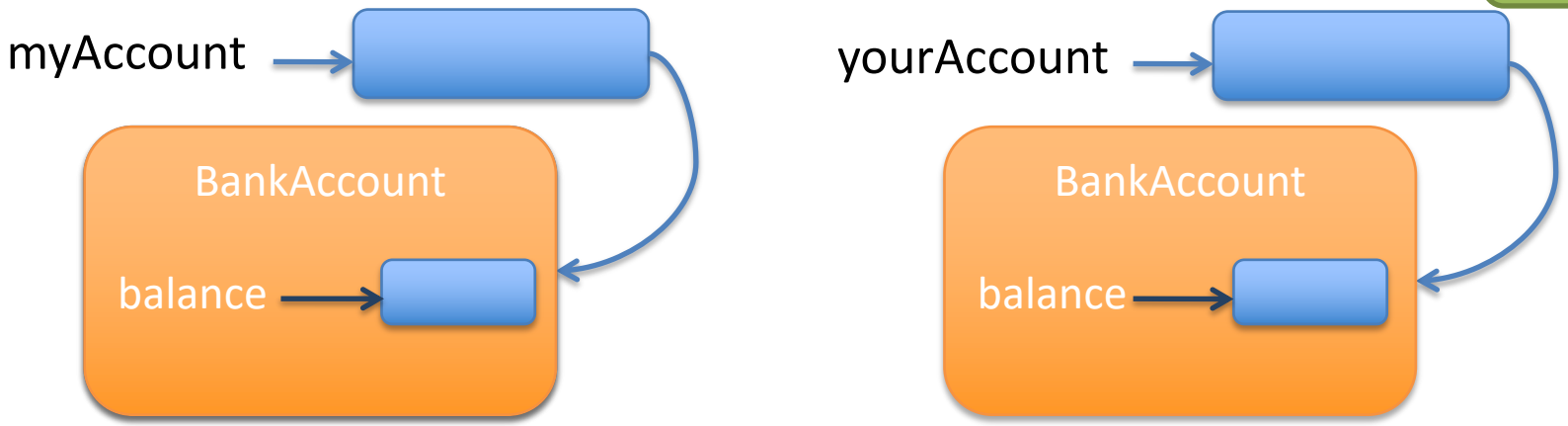
Objects need explicit copyin

Needs to be defined!

```
BankAccount myAccount = new BankAccount(100.0);
BankAccount yourAccount = myAccount.clone();

System.out.println(yourAccount.getBalance());

yourAccount.withdraw(50.0);
System.out.println(myAccount.getBalance());
```

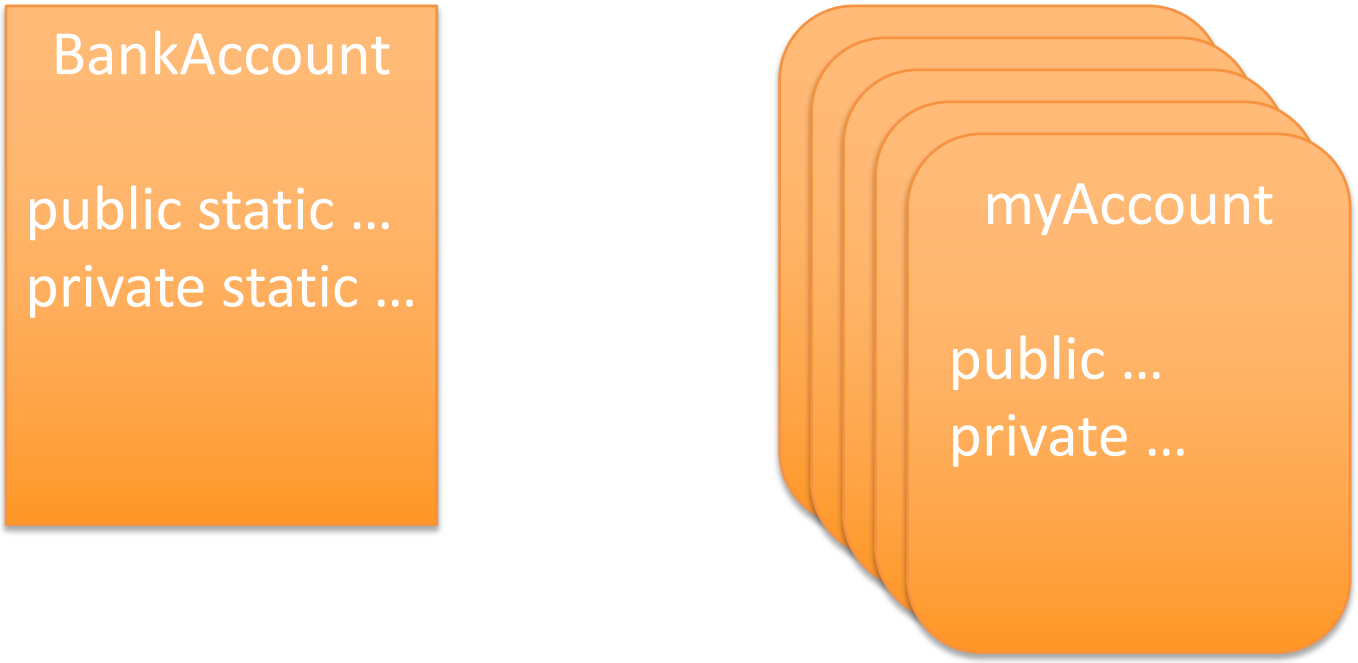
100

100



static

binds variables and methods to the class instead of the instances/objects of the class



Class variables

= variables bound to a class

declares as `static`

- only one per class exists (vs. one per object for instance variables)
- public class variables are referenced using `<nameOfClass>.<variableName>`
- can be constants, declared as `final`

Class methods

= methods bound to a class

declared as `static`

- not invoked on an object
- public class methods are invoked as

```
<nameOfClass>.<methodName>(<parametervalues)
```

Example: Math class

Class variables: actually constants

- Math.PI
- Math.E

Class methods

- Math.abs()
- Math.max()
- Math.random()
- …

Class methods on BankAccount

```
private static double interestRate = 0.02;

public static void setInterestRate(double newRate) {
    interestRate = newRate;
}
```

```
public double getInterest() {
    return balance*interestRate;
}
```

```
public static void main(String[] args)
```

= class method invoked by command line

```
wopr: java$ java BankAccount
```

not every Class is used to start a program

- use the main class to show how a class can/must be used and to test the functionality of the class and its instances
- = Unit Testing: verifying the correctness of a single class in isolation

Object Oriented Programming

All about design

- domain model
- implementation model

1. Find the objects/classes needed for the application
2. Assign responsibilities to the best suited class
3. Implement those responsibilities
4. Test the implementation

Reusable Classes

= foundation of OO-programming

"Whenever possible, steal (re-use) code!"

Java has many re-usable classes

- String
- Scanner
- Math
- System
- …

ArrayList

= object that holds a list of one type of objects

– generic class ArrayList<Type>

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

– can grow and shrink as needed

```
names.add("Bart");
names.add("Lisa");
names.add("Maggie");
```

ArrayList (2)

- get values by index

```
String name = names.get(2);
```

- set values by index

```
names.set(2, "Mulhouse");
```

- remove values by index

```
names.remove(2);
```

- length of ArrayList is available as instance method

```
int length = names.size();
```

- automatic conversion to formatted string

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

[Bart , Lisa]

ArrayList (3)

- can be used for basic types through “auto-boxing”

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

for (int i = 0; i < 10; i++) {
    squares.add(i * i);
}
```

Auto-boxing

- Integer, Double, … are classes representing int, double, … values
- Auto-boxing puts e.g. int-values into Integer classes and gets them out again automatically

Summary

- Objects
  - instance variables
  - instance methods
  - constructors
- Object references
- Classes
  - class variables
  - class methods
- OO programming
  - Re-usable classes, e.g. ArrayLists

On the project

- Only code will not be sufficient
  - Provide illustrative support for your code
    - Comments + descriptions
    - Flowcharts
    - Example runs/traces/logs
    - …
- Pay attention to program structure
  - Group lines of code into meaningful methods
  - Use methods and variables with informative names
  - Use indentation to show structure

Monday in two weeks: Classroom Trial Exam

Like the real one, it'll be more fun when prepared …

- Browse through the slides to refresh your memory. The effort will be useful for next week already.
- If you have the time, read through the book again: chapters 1-6 + 13

  (any topics covered in the book, but not in the lectures/slides will **not** be on the exam, but can still be very interesting)

Tuesday in two weeks

Programming competition

- There will be prices!
- Wear running shoes!

Question asking possibility