CS 171: Introduction to Computer Science II

Methods, OO, Inheritance

Li Xiong

Announcement

- Eclipse/debugging lab
  - 1/30, Monday, 5-6pm, E308
- Hw1
  - To be assigned 1/31, Tuesday
  - Due 2/7, Tuesday

Roadmap

- Review
  - Types, variables, expressions
  - Control flows
  - Methods
  - OO and Inheritance
- Next lecture
  - Arrays and binary search

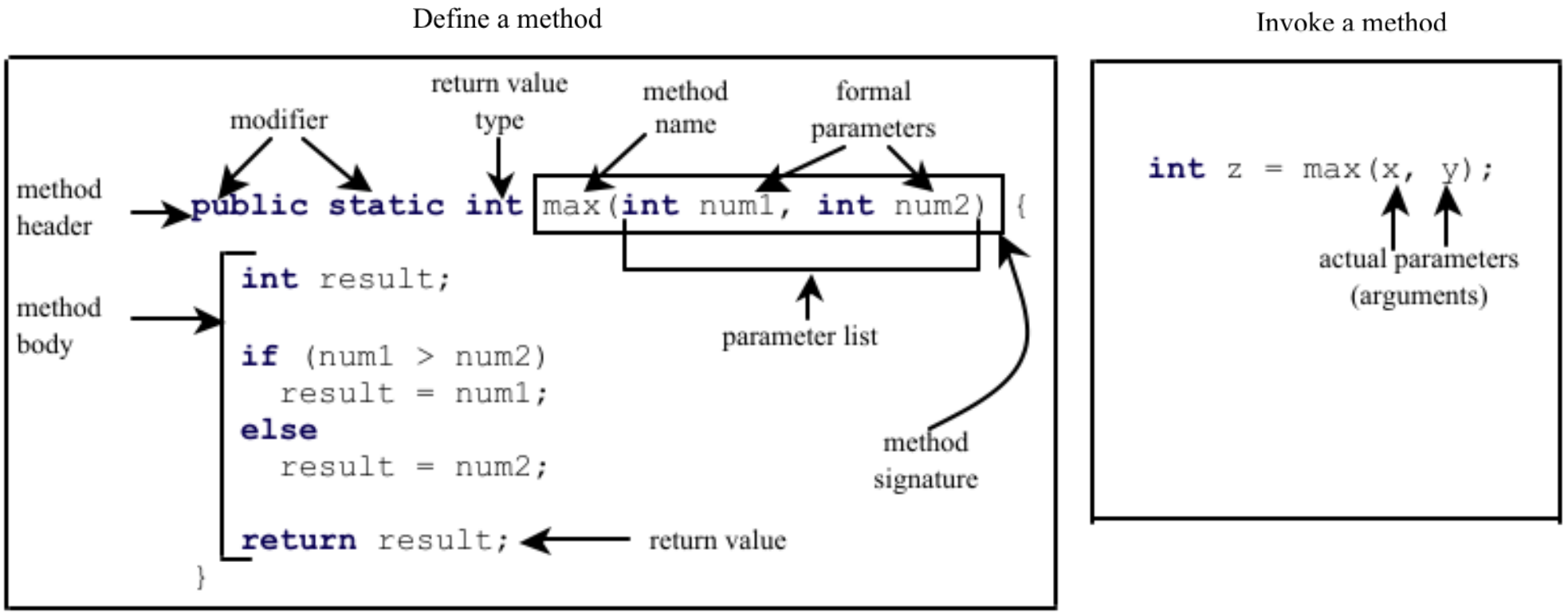
Defining and Using Methods

- Define a method – give a definition of what the method is to do

```
modifier returnType methodName(list of parameters) {
collection of statements;
}
```

- Call or invoke a method – use a method

```
methodName(list of parameters)
```



Passing Parameters

- When calling a method, the arguments must match the parameters in order, number, and compatible type

- When invoking a method, the value of the argument is passed to the parameter. The variable itself is not affected. This is referred to as *pass-by-value*.

Mechanics of the Method-Calling Process

1. Evaluate the argument expressions
2. Copy argument value into the corresponding parameter, (allocated in a newly assigned region of memory called a ***stack frame***)
3. Execute body, using the new stack frame for local variables.
4. On a `return` statement, compute the return value and substitutes that value in place of the call.
5. Discard the stack frame for the method and returns to the caller, continuing where it left off.

Sum Example: Call Stack

```
public static void main(String[] args) {
    // 1. evaluate arguments
    System.out.println("sum(1, 10) is: " + sum(1, 10) ); // 1+2+...+10
    System.out.println("sum(25, 30) is: " + sum(25, 30) ); //25+26+...+30
    System.out.println("sum(40, 50) is: " + sum(40, 50) ); //40+41+...+50
}

public static int sum(int start, int end) { // 2. copy args, new SF
    int sum = 0;
    for (int i = start; i <= end; i++) { // 3. execute the body
        sum += i;
    }
    return sum; //4. return the value, and discard stack frame
}
```

Trace Method Invocation

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

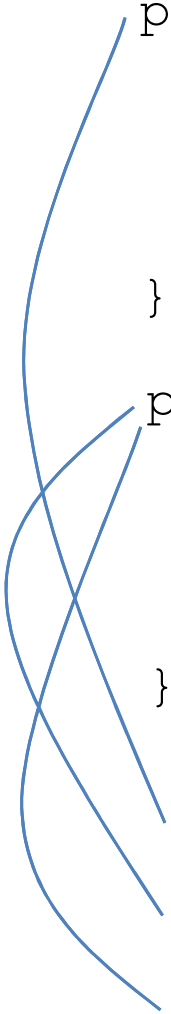
```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Overloading Methods

```
public static int max(int num1, int num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}


 max(1, 3);

 max(1.0, 3.0);

 max(1.0, 3);
```

Overloading Methods

```
public static int max(int num1, int num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}


max(1, 3);

max(1.0, 3.0);

max(1.0, 3);
```



Overloading methods

- Method overloading: multiple methods can have the same name but different parameter lists
- Compiler determines which method is used based on the method signature (method name and parameters)
  - Early binding

Roadmap

- Java Review
  - Types, variables, assignments, expressions
  - Control flow statements
  - Methods
  - OO and Inheritance

Objects and Classes

- Object: entity that you can manipulate in your programs
  - Data fields: state of an object
  - Methods: instructions that accesses or modifies the object
- Class: construct that defines objects of the same type (set of objects with the same behaviour)
  - Definition of data fields: properties of defined objects
  - Definition of methods: behaviours of defined objects
    - Constructors are special type of methods used to construct and initialize objects from the class

Example

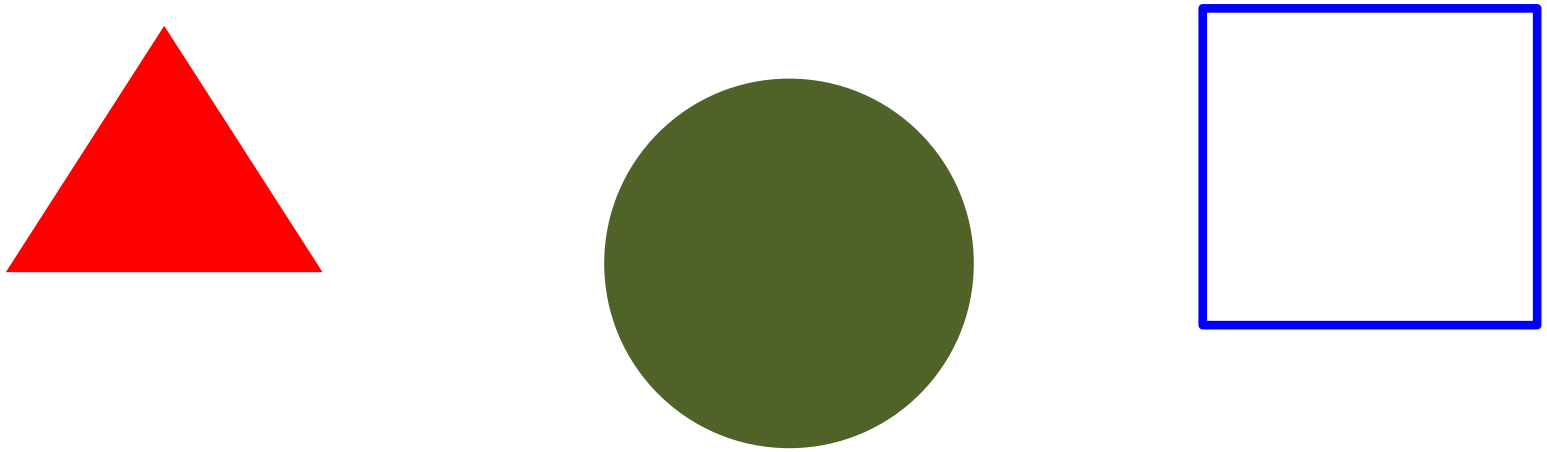
```
public class Employee
{
   private String name; // name of the employee

   public Employee (String n) { name = n; }
   public Employee () { name = "Unknown"; }

   public String getName() { return name; }
   public String toString() { return name;}
}

public class EmployeeTester
{
   public static void main(String[] args) {
        Employee e = new Employee(“The Best Employee”);
        System.out.println(e);
   }
}
```

Have you heard about the object-oriented way to become wealthy?

Inheritance

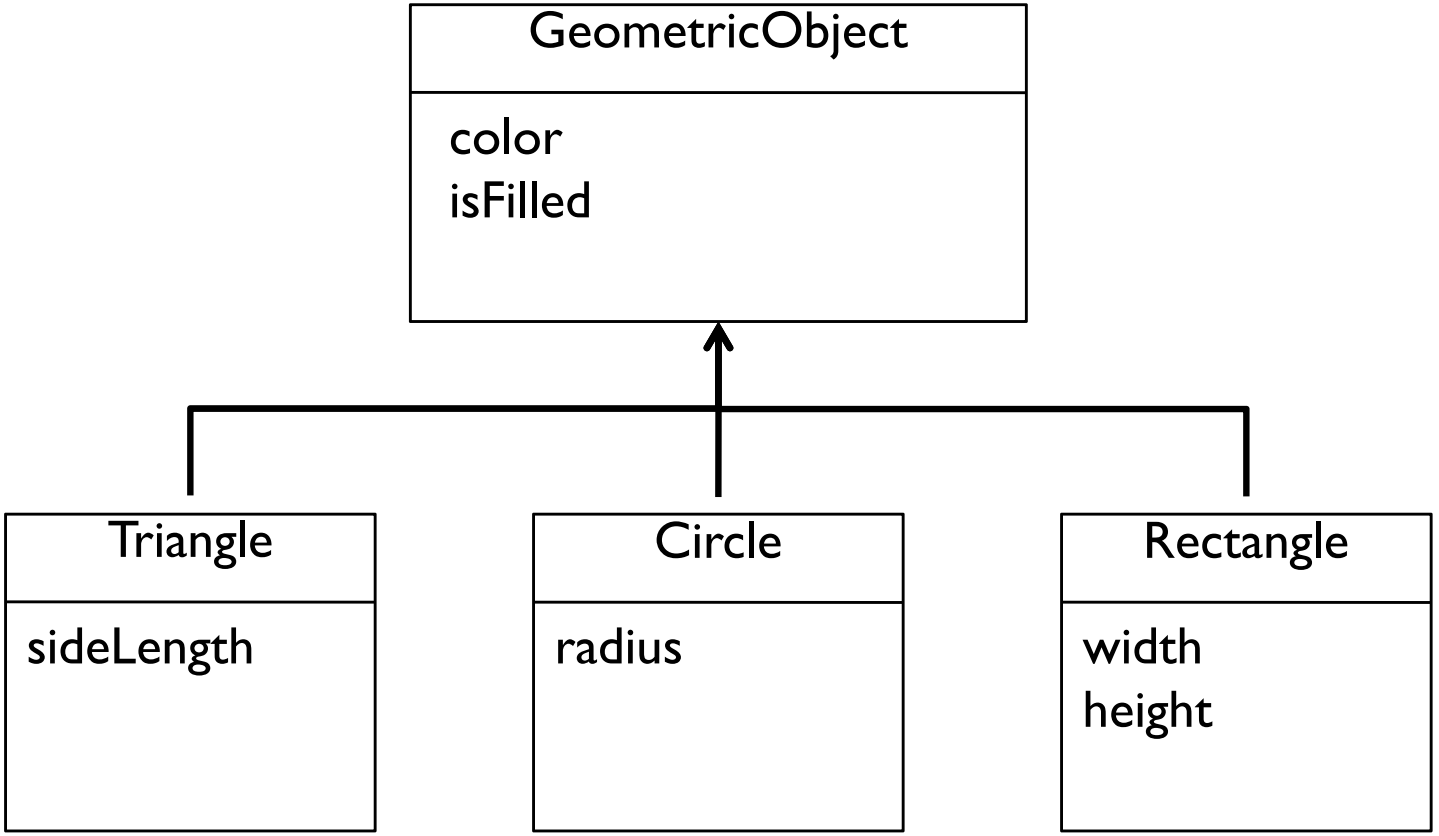
- Different types of employees
  - Hourly employee
  - Salaried employee
  - Volunteer
- What features are common for all employees?
- What features are specific?

Inheritance



- What features are common for all the shapes?
- What features are specific to:
  - Triangle?
  - Circle?
  - Rectangle

Inheritance - idea



`extends` keyword

- Use extends keyword to tell that one class inherits from other class

```
public class GeometricObject {
   public Color color;
   public boolean isFilled;
}

public class Circle extends GeometricObject {
   public double radius;
}
```

Inheritance

- A subclass inherits all fields and methods from the superclass
- A subclass can also:
  - Add new fields
  - Add new methods
  - Override the methods of the superclass

- Superclass's constructor are not inherited
  - Unlike fields and methods
- They are invoked explicitly or implicitly

Using the Keyword super

- `super` refers to the superclass
- This keyword can be used in few ways:
  - To call a superclass constructor
  - To call a superclass method
  - To access a superclass public data field

Invoking Superclass Constructor

- Superclasses' constructors can be invoked from subclasses' constructors explicitly
  - Use the keyword super to call the superclass constructor
  - It must appear first in the constructor
- If no superclass constructor is explicitly invoked, the compiler puts super() as the first statement in the constructor

```
public class Employee
{
   private String name; // name of the employee

   public Employee (String n) { name = n; }
   public Employee () { name = "Unknown"; }

   public String getName() { return name; }
   public String toString() { return name;}
}

public class SalariedEmployee extends Employee
{
   private double weeklySalary;

   public SalariedEmployee(String n; double salary) {
        super(n);
        weeklySalary = salary;
   }

   public double earnings() {
        return weeklySalary;
   }
}
```

Calling Superclass Methods

- super can be used to call method from superclass

```
public class Employee
{
   private String name; // name of the employee

   public Employee (String n) { name = n; }
   public Employee () { name = "Unknown"; }

   public String getName() { return name; }
   public String toString() { return name;}
}

public class SalariedEmployee extends Employee
{
   // …

   public double printName() {
        System.out.println(super.getName());
   }
}
```

Overriding Methods in the Superclass

- Subclass can modify the implementation of a method defined in the superclass
- *Method overriding*
- A private method cannot be overridden, because it is not accessible outside its own class

```
public class Circle extends GeometricObject {
  // Other methods are omitted


  /** Override the toString method defined in GeometricObject */
  public String toString() {
    return super.toString() + "\nradius is " + radius;
  }
}
```

Overriding vs. Overloading

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
  }
}

class B {
  public void p(int i) {
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
  }
}

class B {
  public void p(int i) {
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

```
public class Employee
{
   private String name; // name of the employee

   public Employee (String n) { name = n; }
   public Employee () { name = "Unknown"; }

   public String getName() { return name; }
   public String toString() { return name;}
}

public class SalariedEmployee extends Employee
{
   private double weeklySalary;

   public SalariedEmployee(String n; double salary) {
        super(n);
        weeklySalary = salary;
   }

   public double earnings() {
        return weeklySalary;
   }
}
```

```
public class Employee
{
   private String name; // name of the employee

   public Employee (String n) { name = n; }
   public Employee () { name = "Unknown"; }

   public String getName() { return name; }
   public String toString() { return name;}
}

public class SalariedEmployee extends Employee
{
   // Other methods omitted
   public String toString() {
        return name + ", " + earnings();  // wrong
   }
}
```

```
public class Employee
{
   private String name; // name of the employee

   public Employee (String n) { name = n; }
   public Employee () { name = "Unknown"; }

   public String getName() { return name; }
   public String toString() { return name;}
}

public class SalariedEmployee extends Employee
{
   // Other methods omitted
   public String toString() {
        return getName() + ", " + earnings();  // correct
   }
}
```

```
public class Employee
{
   private String name; // name of the employee
   // other methods omitted
   public String getName() { return name; }
   public String toString() { return name;}
}

public class SalariedEmployee extends Employee
{
   // Other methods omitted
   public String toString() {
       return getName() + ", " + earnings();  // correct
   }
}

public class EmployeeTester {
    public static void main(String[] args) {
        Employee e = new SalariedEmployee("BestEmployee", 2000);
        System.out.println(e);
    }
}
```

Converting Between Subclass and Superclass Types

- Ok to convert subclass reference to superclass reference
- Need cast to convert from a superclass reference to a subclass reference
  - This cast is dangerous: if you are wrong, an exception is thrown
  - Use the `instanceof` operator to test

instanceof

```
 object instanceof TypeName
```

Example:

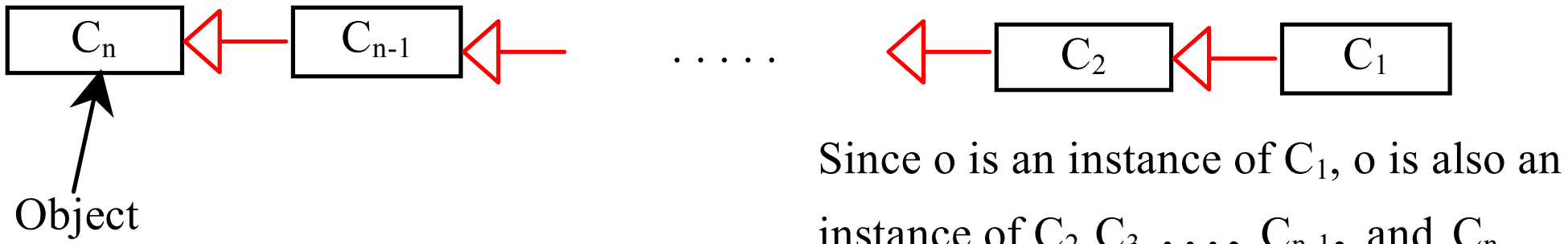
```
 if (anObject instanceof Employee)
{
   Employee e = (Employee) anObject;
   . . .
}
```

Purpose:

To return `true` if the *object* is an instance of *`TypeName`* (or one of its subtypes), and `false` otherwise

Polymorphism and Dynamic Binding

- Method calls are determined by type of actual object, not type of object reference
  - Late binding or dynamic binding (vs. Early binding for overloaded methods)
- Suppose an object o is an instance of classes C1, and C1 is a subclass of C2,..., and Cn-1 is a subclass of Cn (In java, Cn is the Object class), if o invokes a method p, the JVM searches the implementation for the method p in C1, C2, ..., Cn-1 and Cn, until it is found.



```
public class Employee
{
   private String name; // name of the employee
   // other methods omitted
   public String getName() { return name; }
   public String toString() { return name;}
}

public class SalariedEmployee extends Employee
{
   // Other methods omitted
   public String toString() {
       return getName() + ", " + earnings();  // correct
   }
}

public class EmployeeTester {
    public static void main(String[] args) {
        Employee e = new SalariedEmployee("BestEmployee", 2000);
        System.out.println(e);
    }
}
```

Review questions

- Which of the following statements are true?
- Answer: B,C

A. A subclass is a subset of a superclass.

B. A subclass is usually created to contain more functions and more detailed information than its superclass.

C. "class A extends B" means A is a subclass of B.

D. "class A extends B" means B is a subclass of A.

Review questions

- Which of the following statements are true?
- Answer: A,D

  A. A method can be overloaded in the same class.

  B. A method can be overridden in the same class.

  C. If a method overloads another method, these two methods must have the same signature.

  D. If a method overrides another method, these two methods must have the same signature.

Another Example: Bank Accounts

- Bank Account
  - getBalance
  - deposit
  - withdraw
  - transfer
- Savings Account
  - Earns interest that compounds monthly
- Checking account
  - no interest
  - small number of free transactions per month, additional transactions are charged a small fee

`CheckingAccount` Class

- Instance fields:
  - `balance` (inherited from `BankAccount`)
  - `transactionCount` (new to `CheckingAccount`)
- Methods:
  - `getBalance()` (inherited from `BankAccount`)
  - `deposit(double amount)` (overrides `BankAccount` method) – need to update the transaction count
  - `withdraw(double amount)` (overrides `BankAccount` method) – need to update the transaction count
  - `deductFees()` (new to CheckingAccount)

Implementing deposit() method

```
public void deposit(double amount)
{
   transactionCount++;
   // How to add amount to balance
   balance = balance + amount;
   //wrong - balance is a private field of the superclass

}
```

Implementing deposit() method

```
public void deposit(double amount)
{
   transactionCount++;
   // How to add amount to balance

   deposit(amount);

   // wrong – infinite method call loop


}
```

Implementing deposit() method

```
public void deposit(double amount)
{
   transactionCount++;
   // How to add amount to balance

   super.deposit(amount);

   // correct - calling the superclass method


}
```

`SavingsAcount` Class

- Savings account: earns interest with a interest rate
- Instance fields:
  - `balance` (inherited from `BankAccount`)
  - InterestRate (new instance field)
- Methods:
  - `getBalance()`
  - `deposit(double amount)`
  - `withdraw(double amount)`
  - `addInterest()` (new to SavingsAccount)

Exercise

- BankAccountTester.java
  - What's the output?
  - Hint: in the `transfer` method, depending on types of `other`, different versions of `withdraw` and `deposit` are called - polymorphism

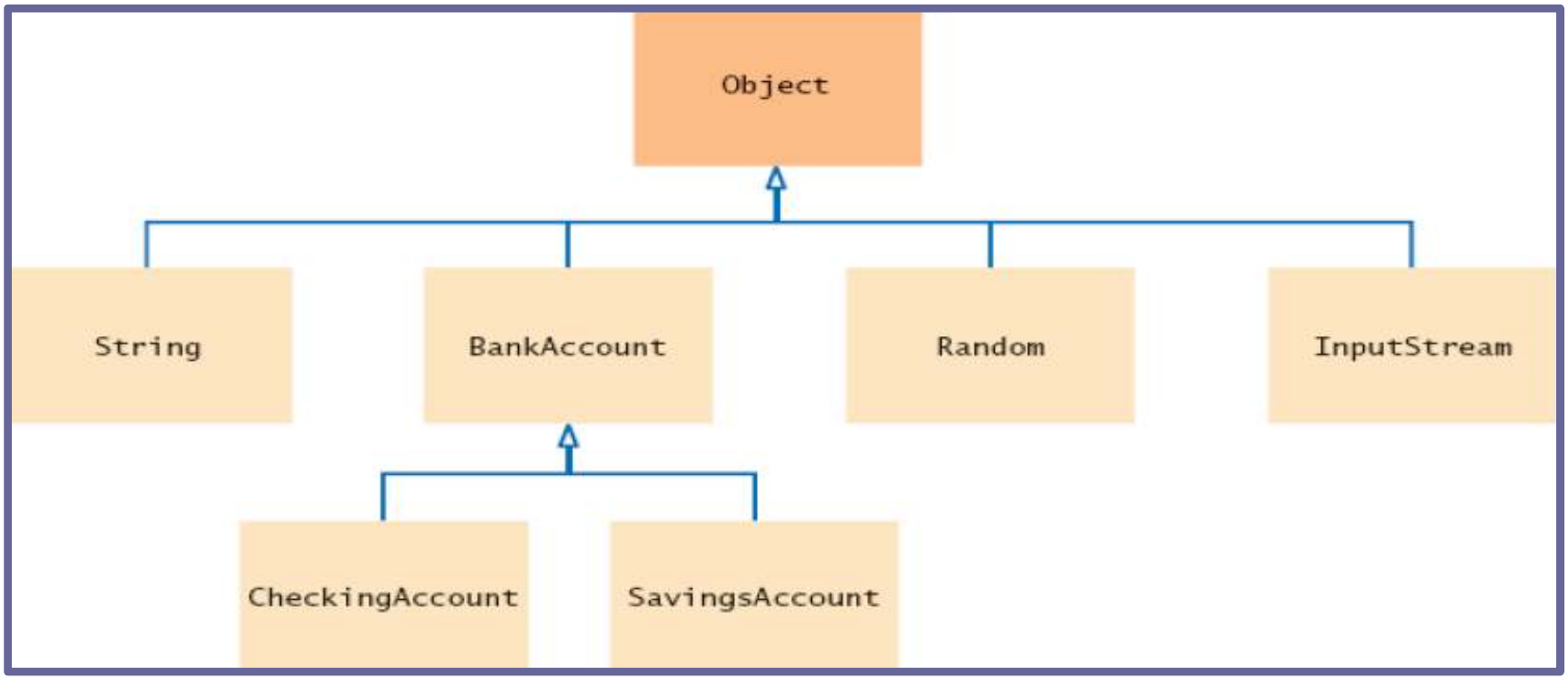
Object: The Cosmic Superclass

- All classes defined without an explicit `extends` clause automatically extend `Object`

```
public class Circle {
  ...
}
```

Equivalent

```
public class Circle extends Object {
  ...
}
```



Object: The Cosmic Superclass

- Most useful methods:
  - String toString()
  - boolean equals(Object otherObject)
- Good idea to override these methods in your classes

Roadmap

- Java Review
  - Types, variables, assignments, expressions
  - Control flow statements
  - Methods
  - OO and Inheritance

- Next lecture
  - Arrays and binary search