Polymorphism

CPSC 1181 – O.O.

Jeremy Hilliker

Summer 2017

Langara.

THE COLLEGE OF HIGHER LEARNING.

Overview

- Type relationships
- Substitution
- instanceof
- Polymorphism
  - Dynamic binding
- Methods on Object
  - .toString
  - .equals

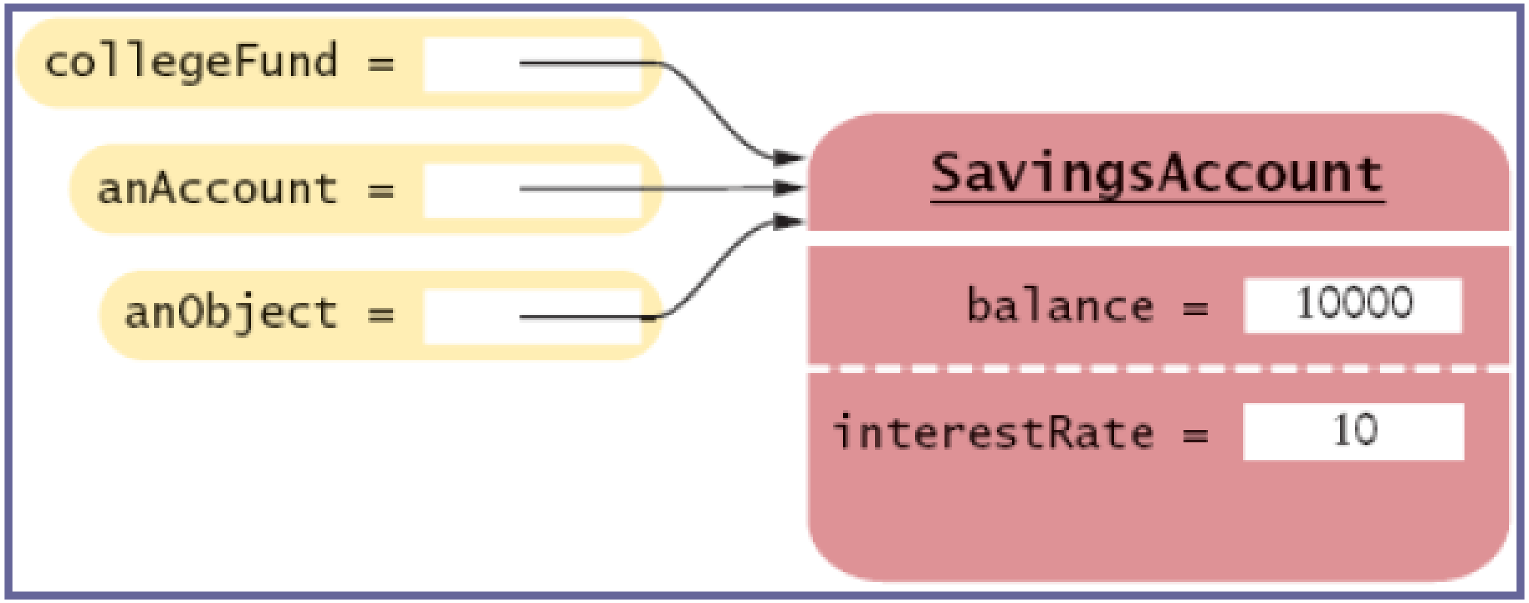
Relationship Between Super- and Sub-Types

- We say that a *subtype* ***is-a*** *specialized* form of its *super-type*
- We know that a *subtype* has *at least* all of the *values* and *behaviours* of its *supertype*
  - Because they are *inherited*

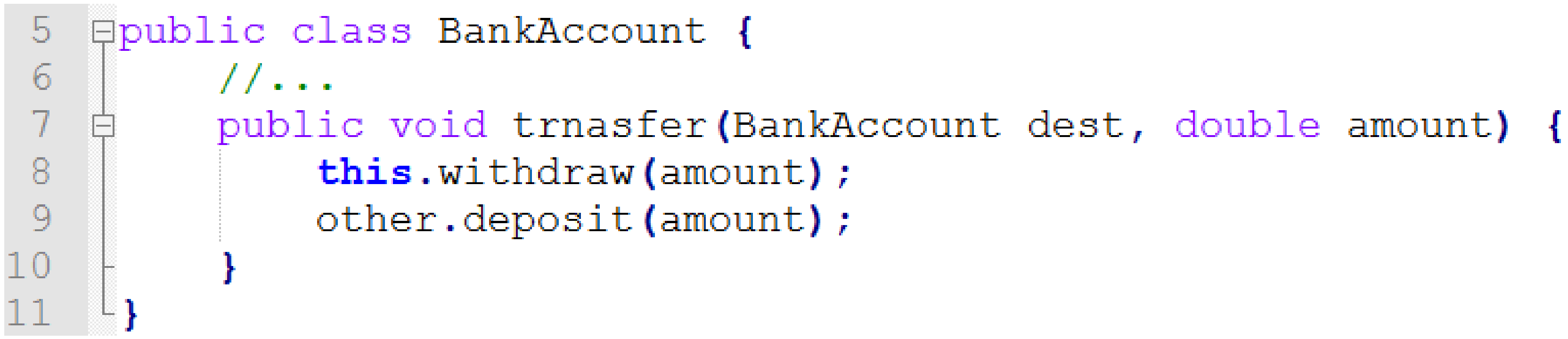
- Therefore, wherever we use a type, we can use a subtype of that type in its place
- This is called "substitution" in the literature
- We can *substitute* a subclass object whenever a superclass object is expected

Substitution

```java
SavingsAccount collegeFund = new SavingsAccount(10);
BankAccount anAccount = collegeFund;
Object anobject = collegeFund;
```



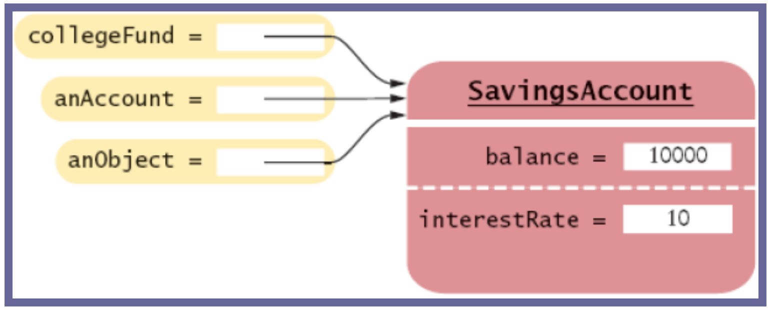
Substitution In Practice:



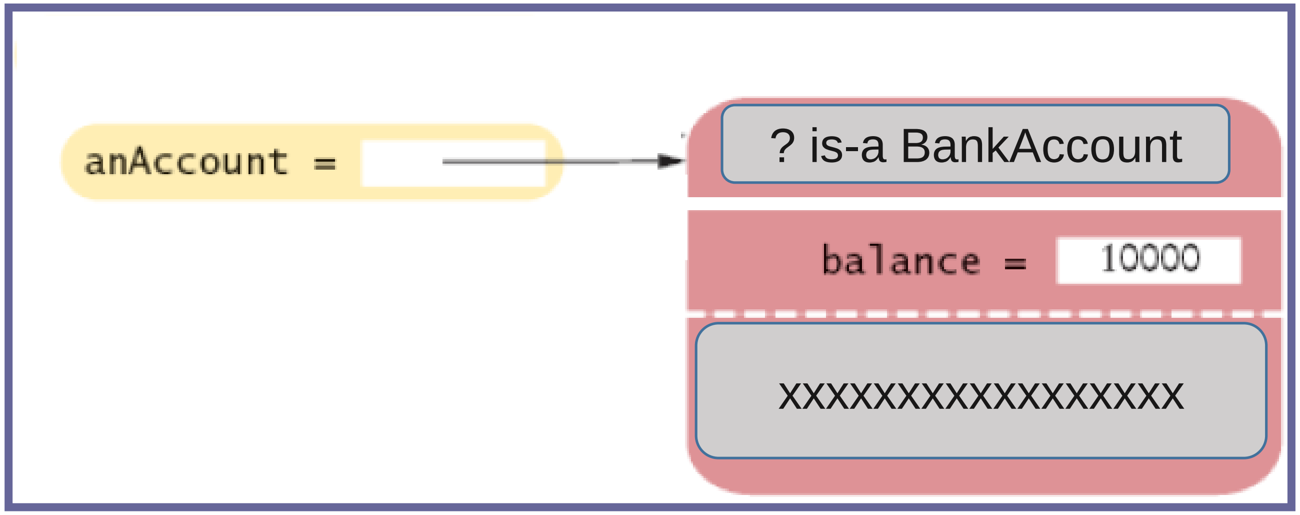
```
public class BankAccount {
        //...
        public void trnasfer(BankAccount dest, double amount) {
                this.withdraw(amount);
                other.deposit(amount);
        }
}
```

- Can assign any *type* of BankAccount to the parameter “other”
  - SavingsAccount
  - CheckingAccount
  - BankAccount
- Code re-use

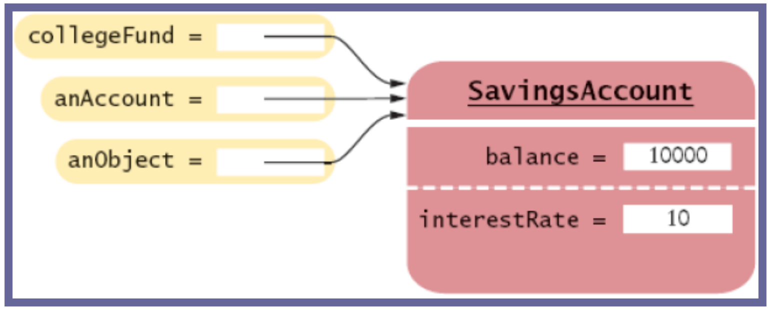
Substitution



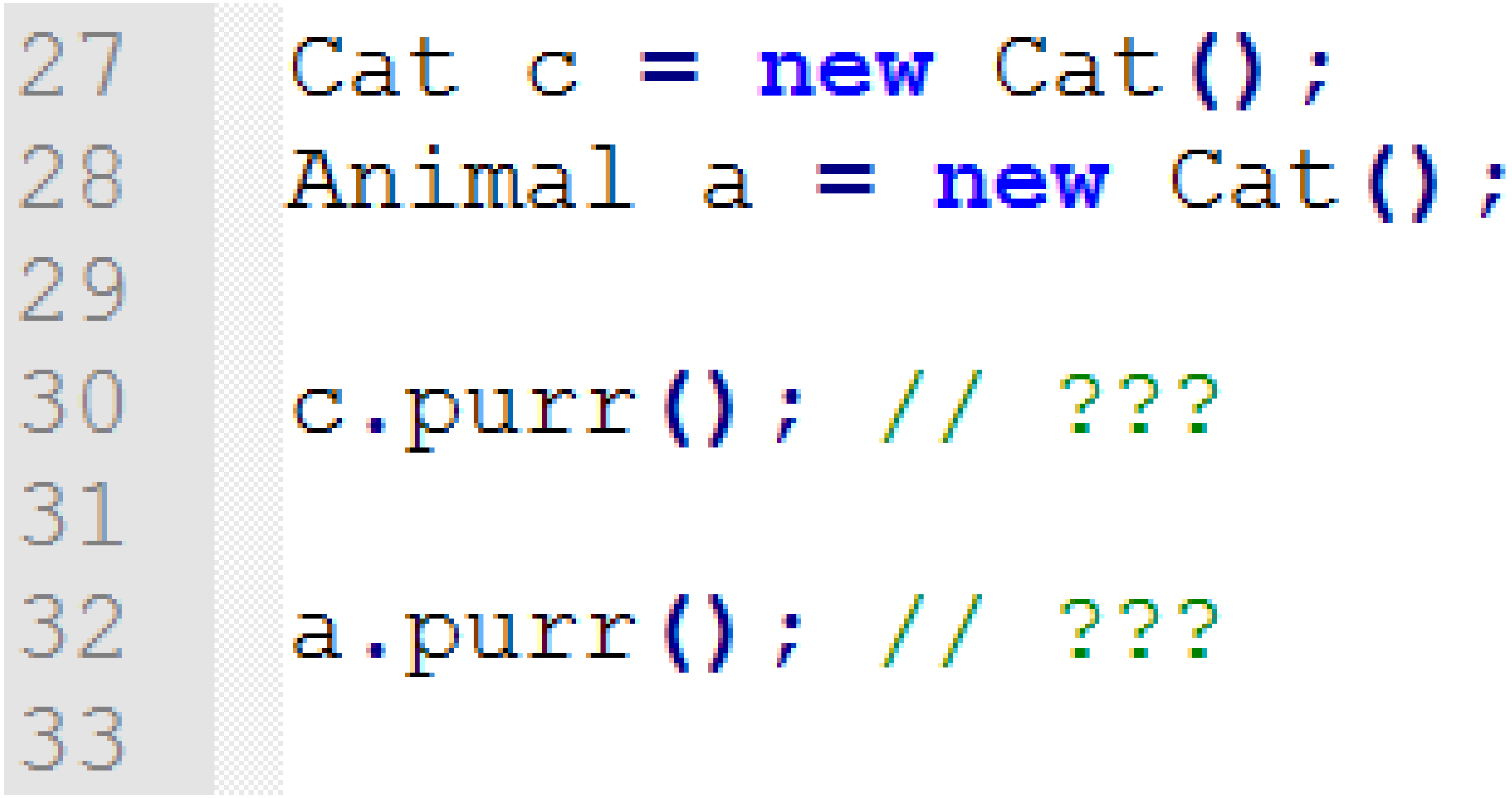
```
SavingsAccount collegeFund = new SavingsAccount(10);
BankAccount anAccount = collegeFund;
Object anobject = collegeFund;

collegeFund.addInterest(); // okay, type "SavingsAccount" has method "addInterest"

anAccount.addInterest(); // compile error.
// anAccount is-a BankAccount
// BankAccount does not han an "addInterest" method!
```



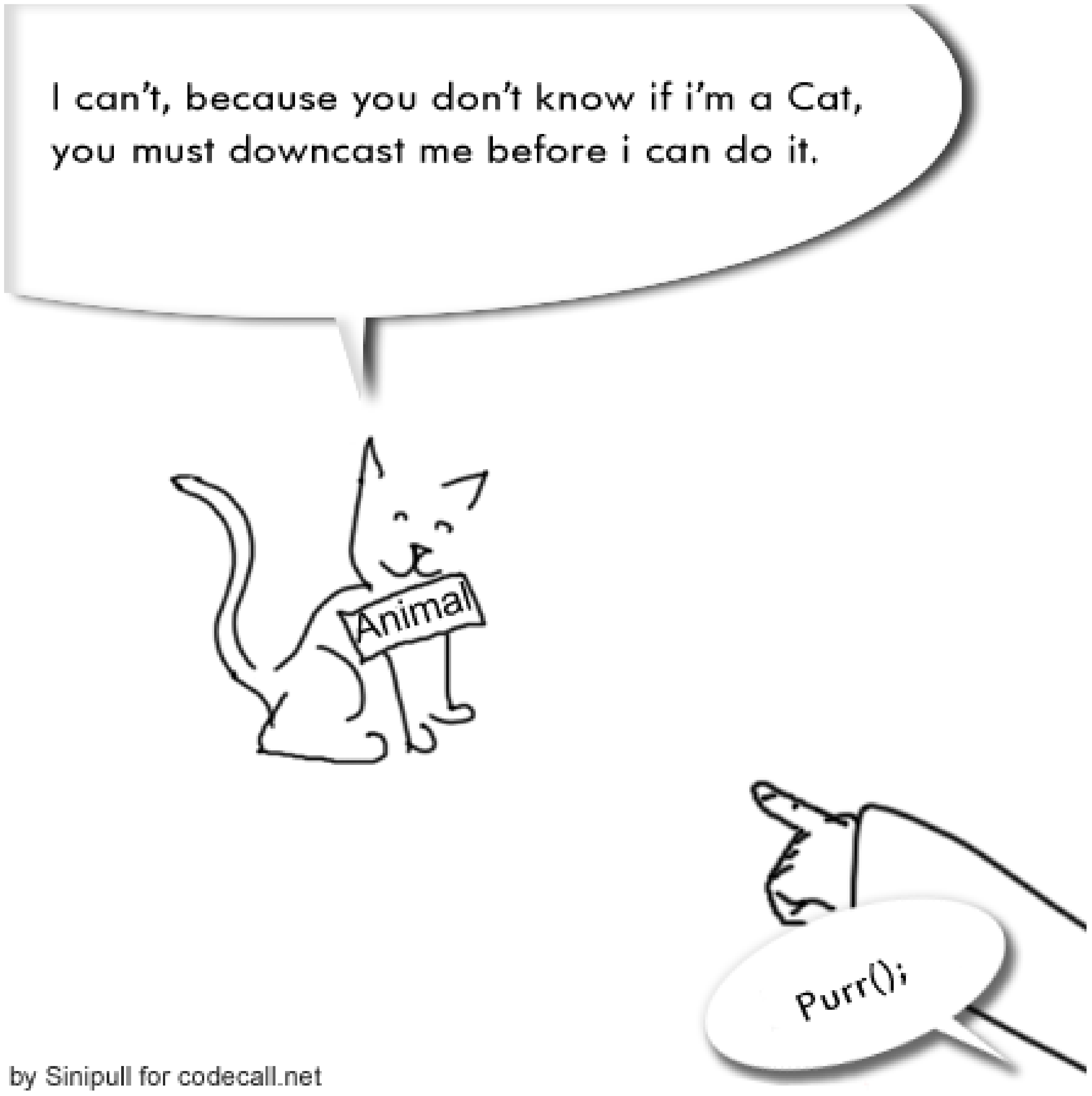
Substitution



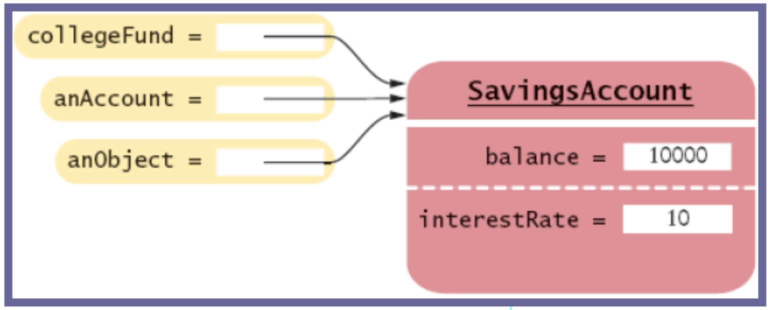
```
SavingsAccount collegeFund = new SavingsAccount(10);
BankAccount anAccount = collegeFund;
Object anobject = collegeFund;

collegeFund.addInterest(); // okay, type "SavingsAccount" has method "addInterest"

anAccount.addInterest(); // compile error.
// anAccount is-a BankAccount
// BankAccount does not han an "addInterest" method!

collegeFund = anAccount; //compile error
// anAccount is-a BankAccount
// BankAccount is-not-a SavingsAccount
```



```java
Cat c = new Cat();
Animal a = new Cat();

c.purr(); // ???

a.purr(); // ???

```


I can't, because you don't know if i'm a Cat, you must downcast me before i can do it.
Animal
Purr();
by Sinipull for codecall.net

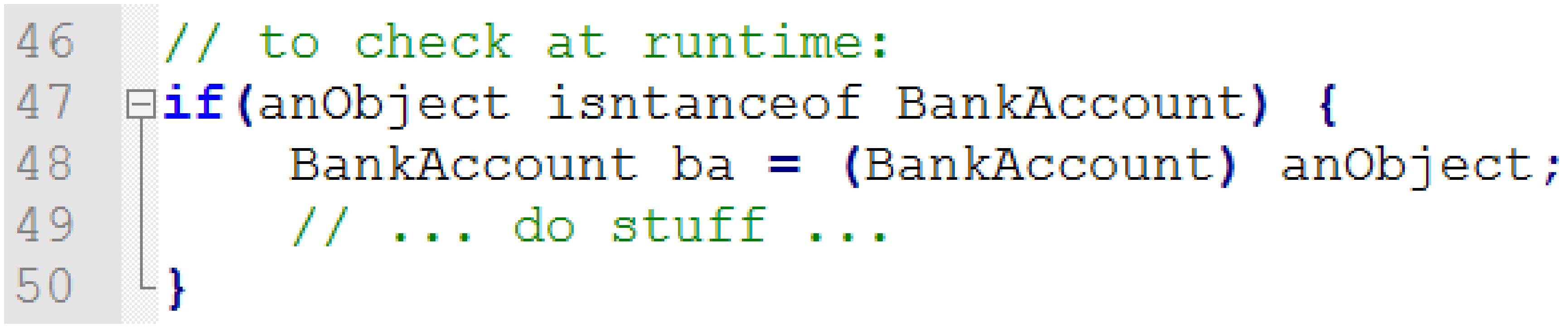
Substitution



```java
SavingsAccount collegeFund = new SavingsAccount(10);
BankAccount anAccount = collegeFund;
Object anobject = collegeFund;

collegeFund.addInterest(); // okay, type "SavingsAccount" has method "addInterest"

anAccount.addInterest(); // compile error.
// anAccount is-a BankAccount
// BankAccount does not han an "addInterest" method!

collegeFund = anAccount; //compile error
// anAccount is-a BankAccount
// BankAccount is-not-a SavingsAccount

collegeFund = (SavingsAccount) anObject; // we super-promise that it's a SavingsAccount
// compiler believes us
// compiles

anObject = new Object();
collegeFund (SavingsAccount) anObject; // we lied
// compiler believes us
// compiles
// SavingsAccount is-a Object, so this cast might be possible
// so the compiler believes us
// but the cast is not possible this time, so it generates a runtime error
// ClassCastExcetption

collegeFund = (SavingsAccount) new ArrayList(); // compile error
// we promise it's a SavingsAccount
// but compiler knows that ArrayList is-not-a SavingsAccount,
// and that SavingsAccount is-not-a ArrayList
// so it doesnt believe us.
```

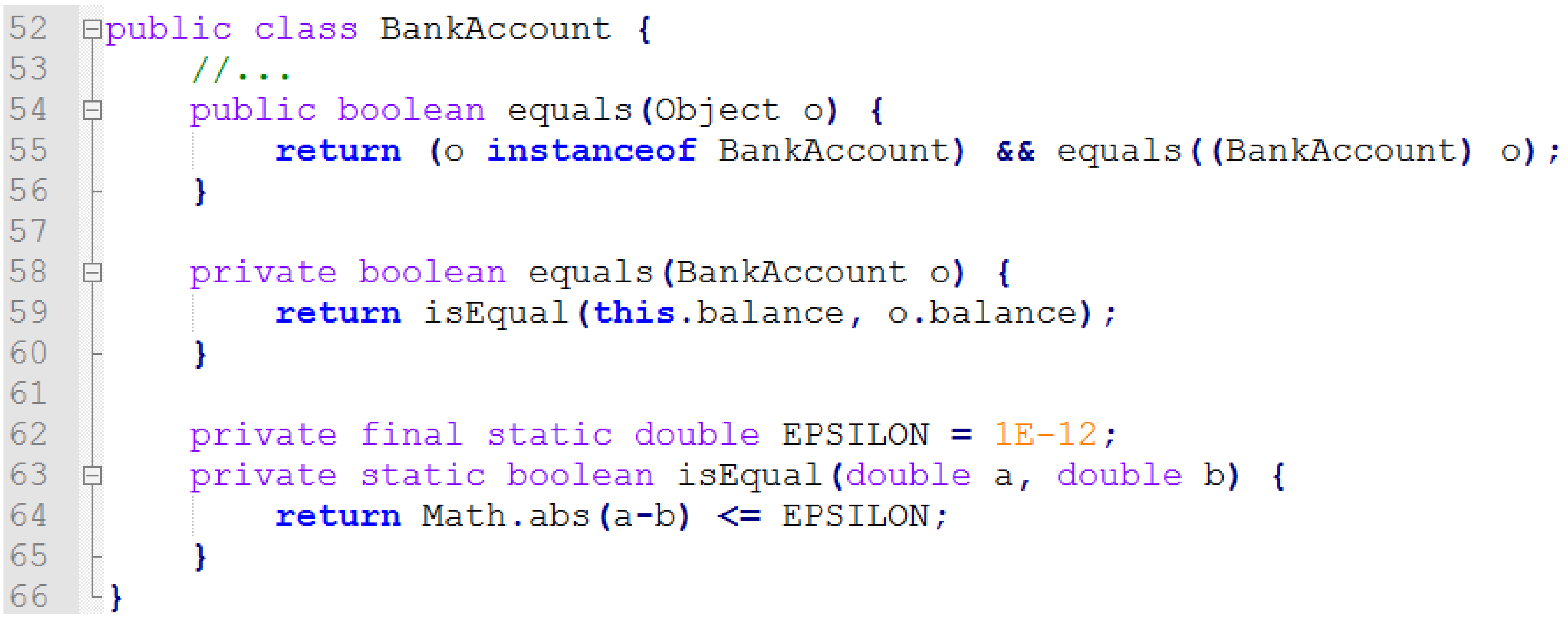
Testing Type: instanceof

- An object’s type can be tested at runtime by
  - ***instanceof***



```java
// to check at runtime:
if(anObject isntanceof BankAccount) {
    BankAccount ba = (BankAccount) anObject;
    // ... do stuff ...
}
```

Ex: instanceof



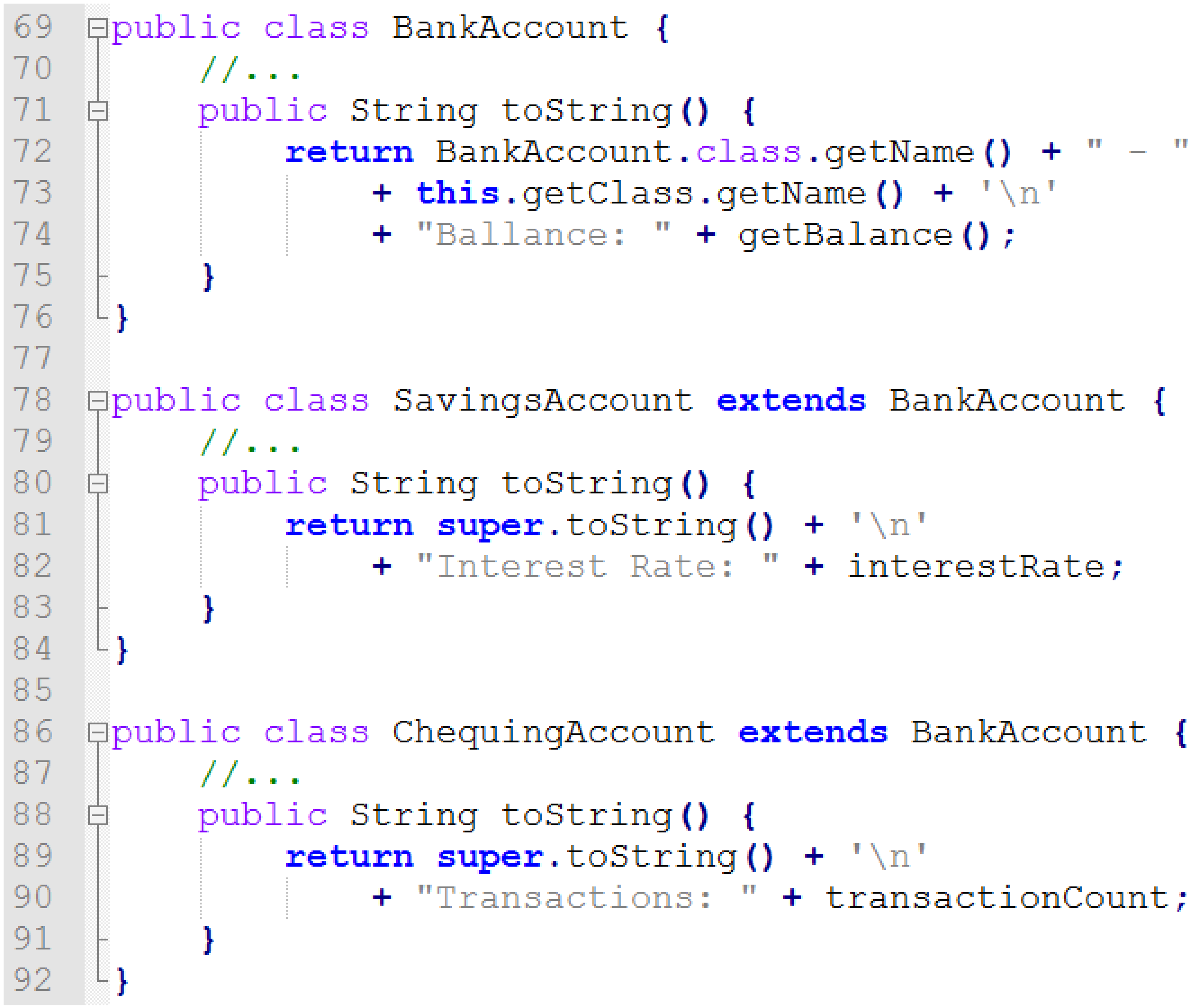
```java
public class BankAccount {
    //...
    public boolean equals(Object o) {
        return (o instanceof BankAccount) && equals((BankAccount) o);
    }

    private boolean equals(BankAccount o) {
        return isEqual(this.balance, o.balance);
    }

    private final static double EPSILON = 1E-12;
    private static boolean isEqual(double a, double b) {
        return Math.abs(a-b) <= EPSILON;
    }
}
```

Polymorphism

- Polymorphism means “having many forms.”
  - A type may declare a behaviour
  - But each subtype may preform that behaviour differently

- Eg: each type *overrides* toString() with its own implementation

Polymorphism:



```java
public class BankAccount {
    //...
    public String toString() {
        return BankAccount.class.getName() + " - "
            + this.getClass.getName() + '\n'
            + "Ballance: " + getBalance();
    }
}

public class SavingsAccount extends BankAccount {
    //...
    public String toString() {
        return super.toString() + '\n'
            + "Interest Rate: " + interestRate;
    }
}

public class ChequingAccount extends BankAccount {
    //...
    public String toString() {
        return super.toString() + '\n'
            + "Transactions: " + transactionCount;
    }
}
```

Polymorphism

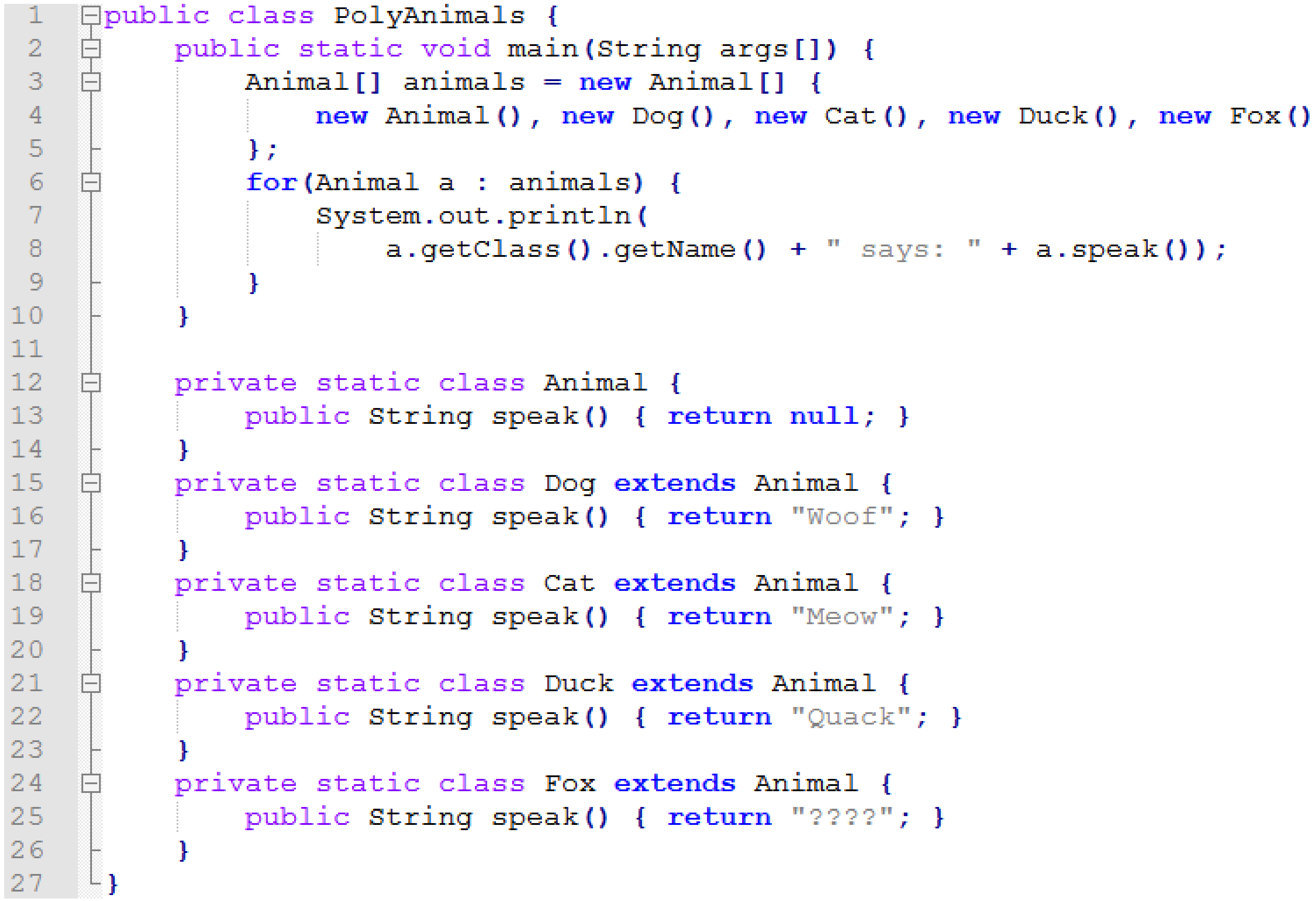
- How does java choose which method to execute?
  - The type declaration may be of a super-type
  - But the object may be of a subtype
- The compiler cannot know the object’s type because of substitution
  - A subtype may have been assigned in place of a super-type
- But the VM does know it

Dynamic Binding

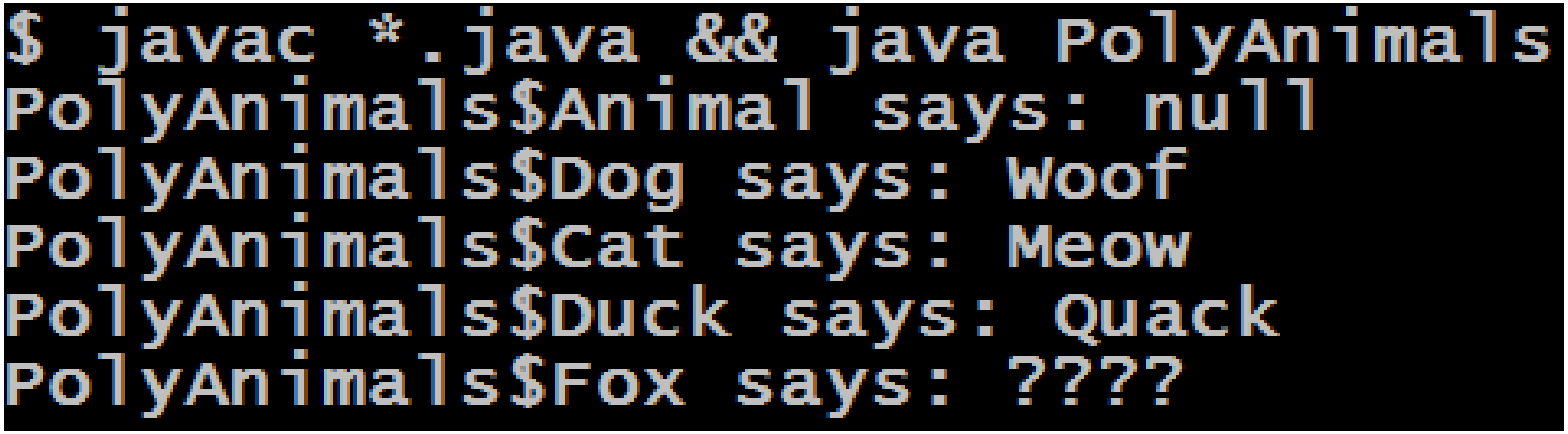
- In practice an object does not have its behaviours stored with it in memory
  - They are stored with the class
- When an instance method is invoked on an object
  - The VM locates the method of the implicit parameter’s class (“this”)
    - Overridden or
    - Inherited
  - This is called ***dynamic binding***
    - Because its done at runtime
    - In c++ it is equivalent to a virtual function

Ex: Polymorphism

Meow!
Woof!
Quack!
Animal
Animal
Animal
Polymorphism
speak();
by Sinipull for codecall.net



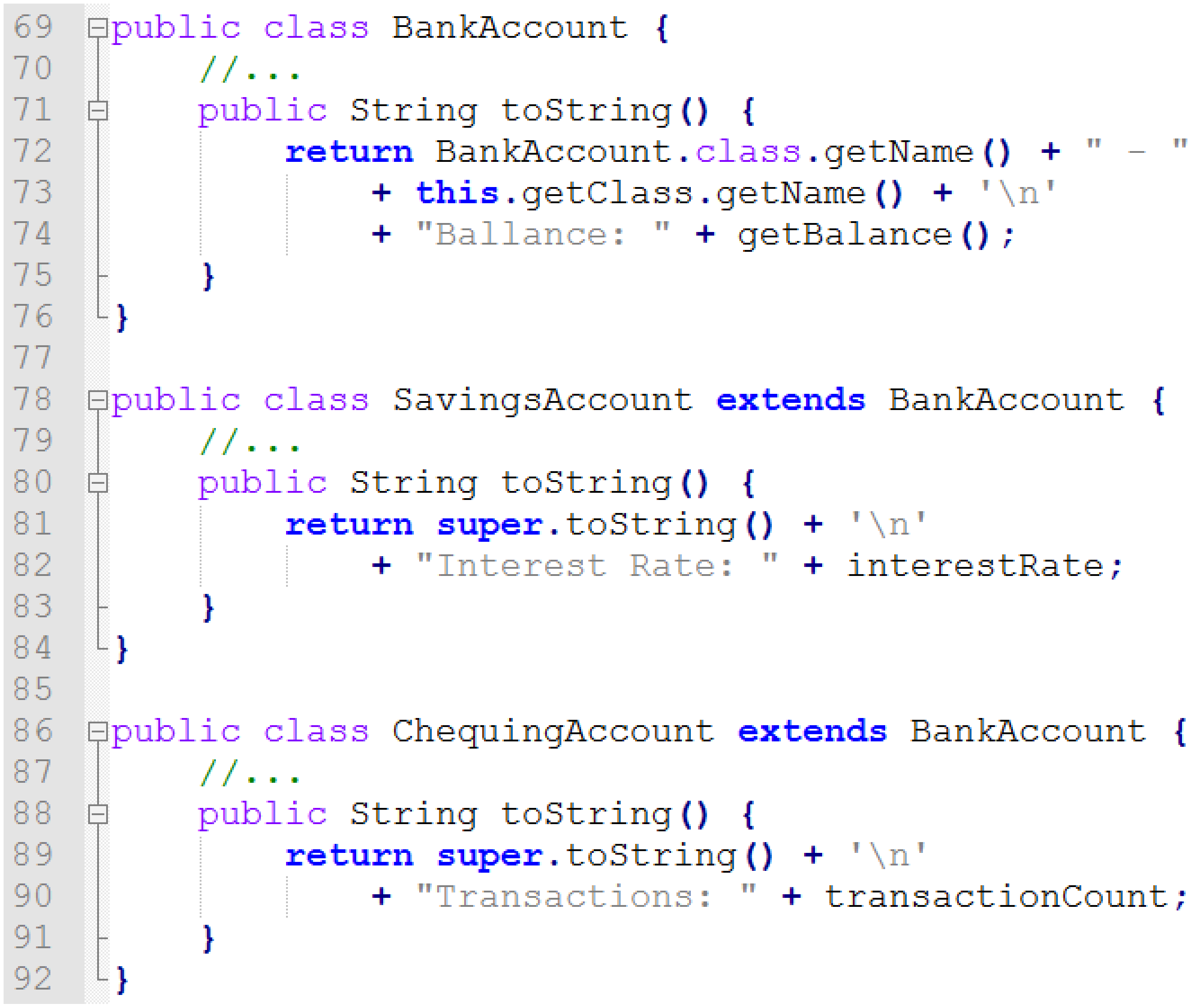
```java
public class PolyAnimals {
    public static void main(String args[]) {
        Animal[] animals = new Animal[] {
            new Animal(), new Dog(), new Cat(), new Duck(), new Fox()
        };
        for(Animal a : animals) {
            System.out.println(
                a.getClass().getName() + " says: " + a.speak());
        }
    }

    private static class Animal {
        public String speak() { return null; }
    }
    private static class Dog extends Animal {
        public String speak() { return "Woof"; }
    }
    private static class Cat extends Animal {
        public String speak() { return "Meow"; }
    }
    private static class Duck extends Animal {
        public String speak() { return "Quack"; }
    }
    private static class Fox extends Animal {
        public String speak() { return "????"; }
    }
}
```



```
$ javac *.java && java PolyAnimals
PolyAnimals$Animal says: null
PolyAnimals$Dog says: Woof
PolyAnimals$Cat says: Meow
PolyAnimals$Duck says: Quack
PolyAnimals$Fox says: ????
```

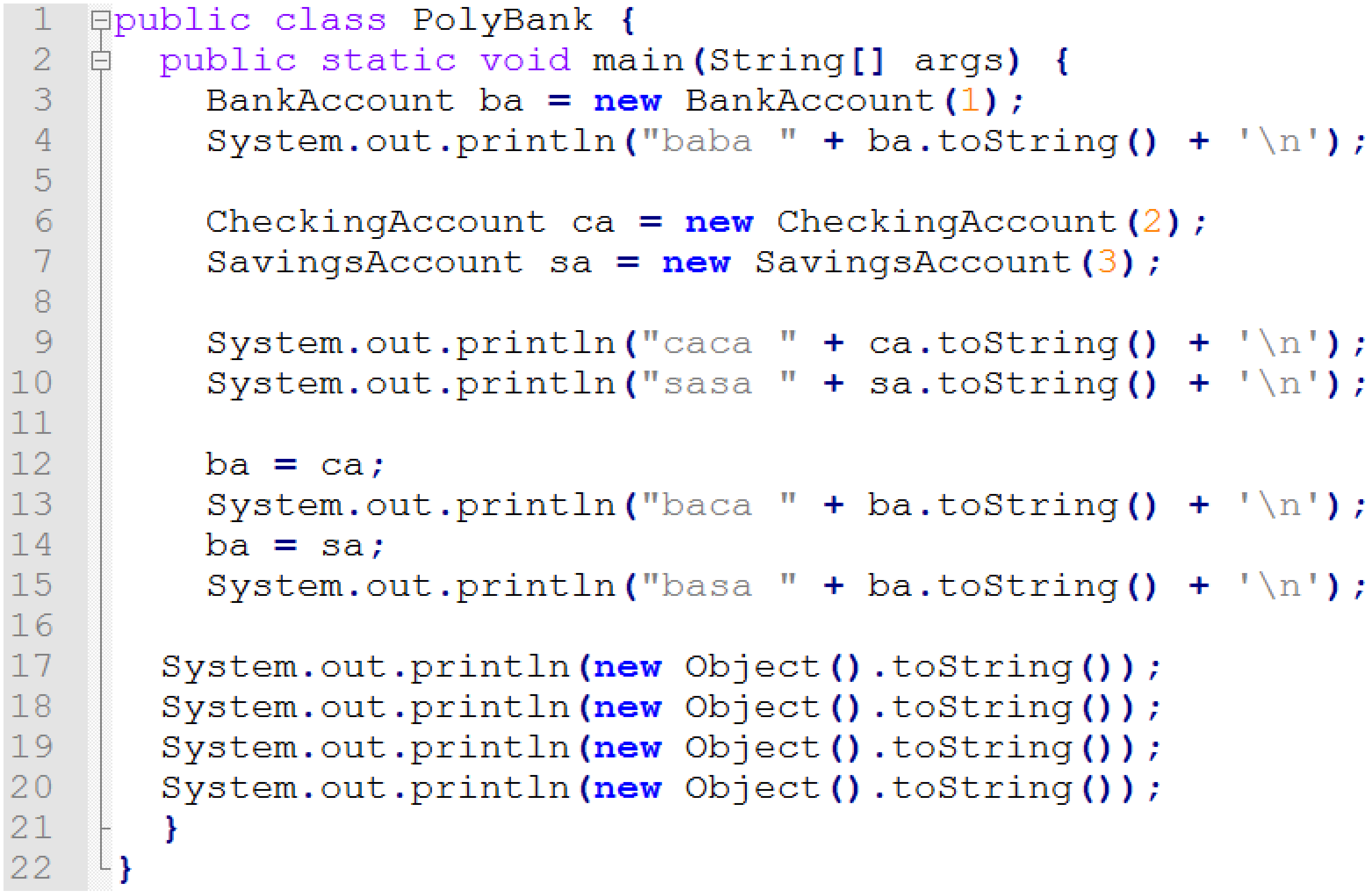
https://www.youtube.com/watch?v=jofNR_WkoCE

Polymorphism:



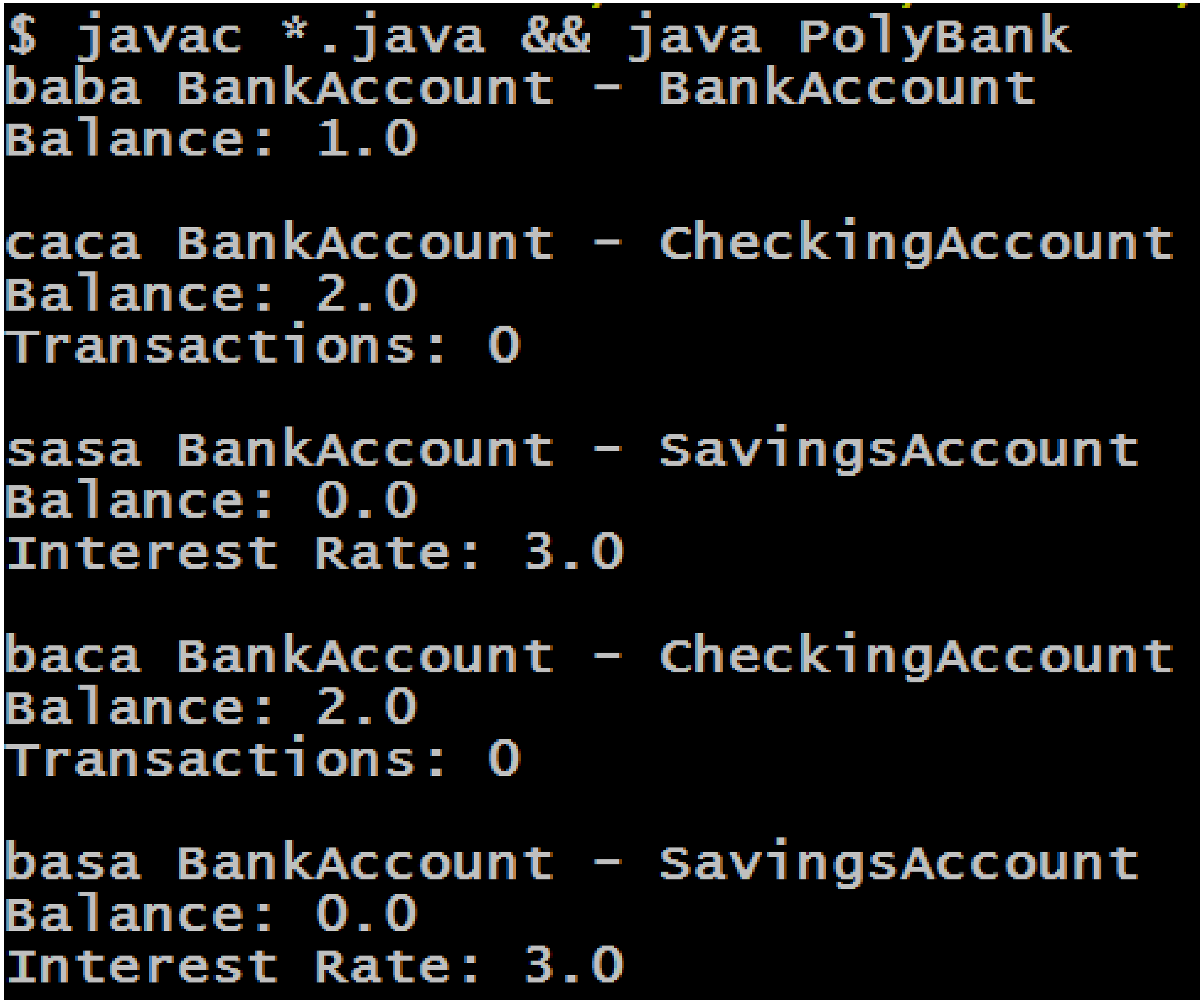
```java
public class BankAccount {
    //...
    public String toString() {
        return BankAccount.class.getName() + " - "
            + this.getClass.getName() + '\n'
            + "Ballance: " + getBalance();
    }
}

public class SavingsAccount extends BankAccount {
    //...
    public String toString() {
        return super.toString() + '\n'
            + "Interest Rate: " + interestRate;
    }
}

public class ChequingAccount extends BankAccount {
    //...
    public String toString() {
        return super.toString() + '\n'
            + "Transactions: " + transactionCount;
    }
}
```

Ex: Polymorphism



```java
public class PolyBank {
  public static void main(String[] args) {
    BankAccount ba = new BankAccount(1);
    System.out.println("baba " + ba.toString() + '\n');

    CheckingAccount ca = new CheckingAccount(2);
    SavingsAccount sa = new SavingsAccount(3);

    System.out.println("caca " + ca.toString() + '\n');
    System.out.println("sasa " + sa.toString() + '\n');

    ba = ca;
    System.out.println("baca " + ba.toString() + '\n');
    ba = sa;
    System.out.println("basa " + ba.toString() + '\n');

  System.out.println(new Object().toString());
  System.out.println(new Object().toString());
  System.out.println(new Object().toString());
  System.out.println(new Object().toString());
  }
}
```

Ex: Polymorphism

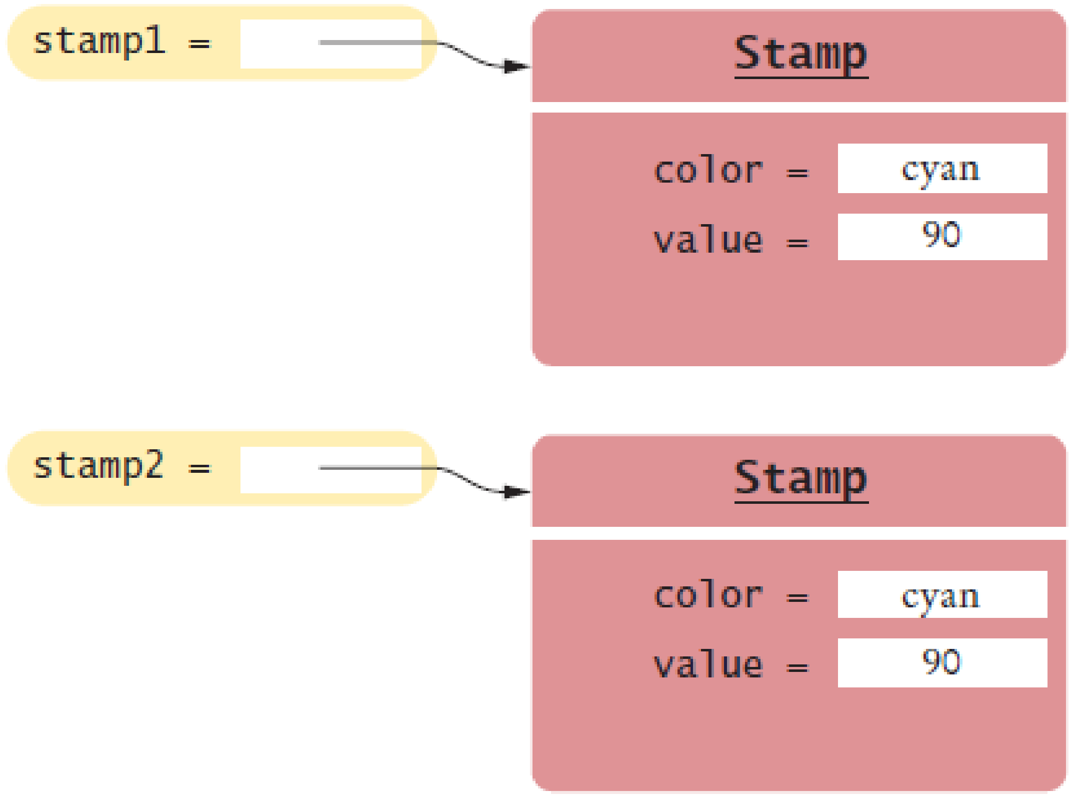


```
$ javac *.java && java PolyBank
baba BankAccount - BankAccount
Balance: 1.0

caca BankAccount - CheckingAccount
Balance: 2.0
Transactions: 0

sasa BankAccount - SavingsAccount
Balance: 0.0
Interest Rate: 3.0

baca BankAccount - CheckingAccount
Balance: 2.0
Transactions: 0

basa BankAccount - SavingsAccount
Balance: 0.0
Interest Rate: 3.0
```

Object

- In java, all object types are subtypes of Object
  - Object is a super-type of all object types
  - So any object can be assigned to type Object
- Some methods defined in Object:
  - toString()
    - Returns a String describing an object
  - equals(Object other)
    - Determines if two objects are semantically equal
      - Contents equal
  - clone()
    - Returns a copy of the object (not used much in practice)
  - hashCode()
    - yields a hash (numerical value) of the object

.equals(Object other)



Note:

.equals(o)

→

.hashCode()

==

o.hasCode()

Recap

- Type relationships
- Substitution
- instanceof
- Polymorphism
  - Dynamic binding
- Methods on Object
  - .toString
  - .equals