Methods

James Brucker

What is a Method?

Programming view: a method is a function. It can return a value or not.

Design view: methods define the behavior of objects. Methods are the way by which objects communicate

```
// deposit some money in an account
public void deposit( double amount ) {
    balance = balance + amount;
}
```

Invoking a Method

To invoke the deposit method, you must use it as a **behavior** of a BankAccount **object**.

Example:

```
BankAccount myAcct = new BankAccount( "Ample Rich" );
Scanner console  = new Scanner( System.in );

// read some deposit data
System.out.print( "Enter deposit amount: " );
long amount = console.nextLong( );

// method:  deposit the money in my account
myAcct.deposit( amount );
```

deposit instance method of a BankAccount object

Static Methods

- Static method is provided by a class, but not part of any object.
- Invoke static methods using the class name:

Math.sqrt(25.0)	sqrt of the Math class
MyClass.main( )	main method of a class
Integer.parseInt("123")	convert String to int
System.exit(0)	exit program.

- Static methods can be used without creating an object from the class.
- This is why "main" is static.

Meaning of Static Methods

- Most static methods are *services* provided by a class.
- Some useful static methods:

```
double length = Math.hypot(3.0, 4.0);
int amount = Integer.parseInt("123");
long now = System.currentTimeMillis();
// create a Calendar object using
// the current Locale information
Calendar cal = Calendar.getInstance();
```

Restrictions of Static Methods

- Static methods cannot directly access the *instance attributes* (object attributes) or *instance methods* of a class.
- Static methods are not polymorphic. You can't implement polymorphism using a static method.
  - Why?
    The compiler "binds" the method call to the method implementation of a particular class *at compile time*.
    [called "static binding"]

main( ) is a static method

- **main( )** is a static (class) method. It cannot call an instance methods unless you create an object first.

```
public class Greeter {
   /** an instance method */
   public String getReply( ) {
      Scanner input = new Scanner(System.in);
      return console.nextLine( );
   }
   public static void main( String [] args ) {
      System.out.print("What's your name? ");
      String name = getReply( ); // ERROR
                    // getReply is not static
   }
```

Instance Methods

- instance methods are the behavior of objects.
- access using the object name:

  "hello there".length( ) length() of the String class

  System.out.printf(...) printf method of "out" object

  x.toString( ) return String form of x
- Instance methods can access the attributes of an object.
- Instance methods *can* call static methods.

Instance Methods

- Instance methods have access to an object's attributes (also called instance variables)
- Instance methods can use the **this** variable.
  this means "this object".

```
public class BankAccount {
    long balance; // balance is an instance variable
    public BankAccount(long abalance) {
        this.balance = abalance;
    }
    public void deposit(long amount) {
        balance += amount;  // same as "this.balance"
    }
```

Writing a Method

A method consists of these parts.

```
// return the maximum of 2 double values
public double max ( double a, double b ) {

    if ( a > b ) return a;
    else return b;
}
```

Parameters

Method Body, with returned value.

Access Control:

`public`
`protected`
`(default)`
`private`

Type of value returned by this method. "void" if nothing is returned.

Writing a Method (2)

This **`max`** method does not access any data other than the parameters, so we *could* make it a **static** method:

```
// return the maximum of 2 double values
public static double max ( double a, double b ) {

    if ( a > b ) return a;
    else return b;
}
```

Declare a "static" method.

It is part of the class, not connected to any object.

Access Control:

`public`, **`protected`**, **`private`**, or default [package access]

Accessing a Method

- **From *inside* of the class**, you can refer to a method using just its name.
- **From *outside* of the class,** you must use a class name (static methods) or object reference (instance method) to call a method.

```
public class MyMath {
   public static double max(double a, double b)
   public static void main( String [] args ) {
      double x = 10.5;
      double y = 10.51;
      // call "max" of MyMath class:
      double r1 = max( x, y );
      // call "max" of Java's Math class:
      double r2 = Math.max( x, y );
}
```

Accessing a Method

- **For instance methods, use an object reference to qualify method access.**

```
public class Bank {
   public static void main( String [] args ) {
      BankAccount a = getAccount("Ample Rich");
      BankAccount b = getAccount("Still Poor");
      // call "withdraw" of object a:
      Money amount = a.withdraw( 100000 );
      // call "deposit" of object b:
      b.deposit( amount );
    }
```

a and b are references to BankAccount objects.

Visibility (Accessibility) of Methods

You control what objects can access an object's methods.

There are **4** choices:

- **private**: method can only be invoked by code in this class.
- **protected**: method can be invoked by other classes in the same package, or by any subclass of this class.
- **public**: method can be invoked by any Java program.
- **default**: can only be invoked by other classes in same package

```
public void deposit( long amount ) {
     /* body of the method */
}
```

Return Value of a Method

- A method may return a value. The type of return value must be declared in the method header.
- A method which doesn't return any value should have a return type of "void".
- In the method body, use "return <expression>".

```
class BankAccount {

  public void deposit(long amount) {
      balance += amount;
  }
  public long getBalance( ) {
      return balance;
}
```

void means this method does not return a value.

Common Method Types

These are examples of common methods.

```
int getValue( )

void setValue( int value )

boolean equals( Object other )

int hashCode( )

String toString( )

int compareTo( MyClass other )
```

Constructor

- A constructor is **not** a method, but the syntax is similar
- A constructor may have parameters.
- A constructor has **<u>no</u>** return value, not even "void".

```
public class BankAccount {
    public BankAccount( ) {
        balance = 0;
        acctName = null;
    }
    public BankAccount(String name) {
        balance = 0;
        acctName = name;
    }
    public void BankAccount(String name, long balance)  {
```

no return value

not a constructor

Review Questions

Identify each method as Static or Instance Method

- console.**nextInt**( ); // console is a Scanner object
- String s = "This is too easy.";

  s.**length**( )
- double angle = Math.toRadian(45);
- System.out.**println**("Print me");
- Double.**parseDouble**("123.45E-12")

Identify all 3 methods. getTime() returns a Date.

- Calendar.**getInstance**( ).**getTime**( ).**getMonth**( )

Interpretation of Static Methods

A static method can be:

- a **service** provided by the class
- a "public utility", like the methods in Math, or
- a way to create objects from the class (or another class). Useful if creating objects is complex.
  - This is called a *Factory Method*.

```
// Calendar.getInstance returns a new Calendar object,
// with the default timezone and locale
// getInstance is a Factory Method for the Calendar class.
Calendar date = Calendar.getInstance( );
```

Static and Instance Methods

- What is wrong here?

```
public class TestProgram {
    public int max( int m, int n ) {
        return (m>n)? m : n ;
    }

    public static void main( String [] args ) {
        int n = 100;
        int m = 200;
        System.out.println("max of m, n is " +max(m,n));
     //  ERROR.  Why?
    }
}
```

Static Methods, Instance Variables

- What is wrong here?

```
public class BankAccount {
   long balance;
   long accountNumber;
   String accountName;
   /** next available account number */
   static long nextAccountNumber = 1;

   pubic static long getNextAccountNumber()
       {
      nextAccountNumber++;
      accountNumber = 0;
      return nextAccountNumber;
   }
```