Defining A Class of Objects

• Defining a class of objects requires that we declare all of the properties of the objects, as well as its methods.

• All items included should be explicitly declared as *private* (available only to functions within the class of objects) or *public* (available and accessible to all methods of all objects of all classes).
Syntax For a Class definition

• The syntax is:

```java
public class classname {
  declarations for properties
  declarations for methods
}
```

Public Methods

• Public methods can be called by any other method whether it is in the same class or in a different class.
• We can now leave off the word static if these methods belong to the individual objects and not to the class itself.
An example of a Class: Grades.java

```java
import java.util.Scanner;

// Grades - A class of course and exam grades
public class Grades {

    // readGrade() - Returns a percentage for a course
    public int readGrade() {
        Scanner keyb = new Scanner(System.in);
        int inputGrade;

        // Keep asking until the user enters a valid percentage grade
        do {
            System.out.println("Enter your grade " +
                              + " (0-100%) ");

            inputGrade = keyb.nextInt();
        } while (inputGrade < 0 || inputGrade > 100);

        // Invariant - inputGrade is a valid percentage grade
        return inputGrade;
    }

    // An example of a Class:
    // Grades.java
```
// printGrade() - Prints the test grade along with the test number
public void printGrade(int examNumber,
                        int outputGrade) {
    System.out.println("Test #" + examNumber + ":	" + outputGrade);
}

// findAverage() - Finds the average of four grades
public double findAverage(int test1,
                           int test2, int test3, int test4) {
    double sum = test1 + test2 + test3 + test4;
    return sum/4;
}

// convertGrade() - Converts a test average into a letter grade based on the formula that 90+ is an A, 80-89 is a B and so on
public char convertGrade(double numberGrade) {
    if (numberGrade >= 90)
        return 'A';
    else if (numberGrade >= 80)
        return 'B';
    else if (numberGrade >= 70)
        return 'C';
    else if (numberGrade >= 60)
        return 'D';
    else
        return 'F';
}
Declaring Objects

- We declare new objects of a user-defined class the same way as we do standard classes:
  
  ```java
  ClassName objectName = new ClassName();
  ```

- Any methods belonging to the object are called the same way as methods of standard classes:
  
  ```java
  objectName.methodName();
  ```

Using the Class – **FindGrades.java**

```java
import java.util.Scanner;

public class FindGrades {
    // Find a student’s course grade based on the test scores
    public static void main(String[] args) {
        Grades grades = new Grades();
        int exam1, exam2, exam3, exam4;
        double meanGrade;
        char letterGrade;

        // Get the test grades
        exam1 = grades.readGrade();
        exam2 = grades.readGrade();
        exam3 = grades.readGrade();
        exam4 = grades.readGrade();
    }
}
```
// Print the grades
printGrades(grades, exam1, exam2, exam3, exam4);
// Find the average and letter grade
meanGrade = grades.findAverage(exam1, exam2, exam3, exam4);
letterGrade = grades.convertGrade(meanGrade);

// Print the results
System.out.println("The average is "+ meanGrade
+ " which is a letter grade of "+ letterGrade);
}

// printGrades() – Print the test scores
public static void printGrades(Grades g, int test1, int test2, int test3, int test4) {
    System.out.println("These are your test scores: ");
g.printGrade(1, test1);
g.printGrade(2, test3);
g.printGrade(3, test3);
g.printGrade(4, test4);
}
}
Defining Properties

- Properties are values that variables associated with a given type of object may have, such as a person’s name (or height or hair color).
- Properties can be declared as independent of any specific method in a class and can then be used by any method in that class.

Defining Properties: An Example

```java
import java.util.Scanner;

public class MyInt {
    public int x;  // A property - accessible by // any other class

    // read() - Reads an integer
    public void read() {
        Scanner keyb = new Scanner(System.in);
        System.out.println("Enter an integer?");
        x = keyb.nextInt();
    }

    // write() - Writes an integer
    public void write() {
        System.out.println("Value is = " + x);
    }
}
```
// changeValue() - Lets you change an integer value
public void changeValue(int y) {
    x = y;
}

Using Private Properties: An Example

import java.util.Scanner;

public class MyInteger {
    private int x;    // A private property - can
    // only be used by the
    // class's own methods

    // read() - An input method
    public void read() {
        Scanner keyb = new Scanner(System.in);
        System.out.println("Enter an integer?");
        x = keyb.nextInt();
    }

    // write() - An output method
    public void write() {
        System.out.println("Value is = " + x);
    }
}
// setX() - a Mutator - it changes a class property
public void setX(int y) {
    x = y;
}

// getX() - an Accessor - it accesses a class property
public int getX() {
    return x;
}

public class Point {  
    // Private data should go first  
    private double x, y;  

    // Start placing the methods HERE  
    // The constructors belong on top  

    // Point() - Default constructor  
    public Point() {
        x = 0;
        y = 0;
    }
}

Example of a Class definition

// Import the libraries that this class needs  
import java.util.Scanner;  
import java.math.*;

public class Point {
    // Private data should go first  
    private double x, y;  

    // Start placing the methods HERE  
    // The constructors belong on top  

    // Point() - Default constructor  
    public Point() {
        x = 0;
        y = 0;
    }
}
// Point() - Initialization constructor
public Point(double initX, double initY) {
    x = initX;
    y = initY;
}

// read() - An input method
// Needs to be able throw an exception
public void read() {
    Scanner keyb = new Scanner(System.in);
    String inString = new String();

    System.out.println("Enter x\t?");
    x = keyb.nextInt();

    System.out.println("Enter y\t?");
    y = keyb.nextInt();
}

// write() - An output method
public void write() {
    System.out.print("( + x + , + y + )");
}

// distance() - This version returns distance from the origin
public double distance() {
    return Math.sqrt(x*x + y*y);
}

// distance() - This version returns the distance from another point
public double distance(Point p) {
    return Math.sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
}
Example: Rewriting the Original Age Program

- Let’s rewrite the program that asked for name and age and then printed these items.
- There are two data items, both of which should be private: name and age.
- There are two procedures, both of which should be public: read and write.

```
import java.util.Scanner;

public class OldGuy {

    // The private properties
    private String name;
    private int age;

    // Mutators - methods that change private properties
    public void setName(String newName) {
        name = newName;
    }
}
```
public void setAge(int newAge) {
    age = newAge;
}

// Accessors - methods that return the values
// of private properties
public String getName() {
    return name;
}

public int getAge() {
    return age;
}

// read() - an input method
public void read() {
    Scanner keyb = new Scanner(System.in);
    String inString = new String();

    // Read in name and then age
    System.out.println("What\'s your name\t?");
    name = keyb.nextLine();
    System.out.println("How old are you\t?");
    age = keyb.nextInt();
}
// write() - an output method
public void write() {
    System.out.println(name + " is " + age
    + " years old.");
}

TestOldGuy.java

public class TestOldGuy {
    public static void main(String[] args) {
        OldGuy me = new OldGuy();
        me.read();
        me.write;
    }
}
Member Functions and Parameters

• Functions belonging to a class can have parameters, including other objects of the same class or different class.
• If you pass as a parameter an object of the same class, you must use the name of the object when specifying its members.
• E.g., y is an item in this object, q.y is an item in object q.

Rewriting age To Include New Members

• Let’s rewrite the class to include the functions older and younger, which return true or false, depending on whether this person is older or younger than the person passed as a parameter.
Added to OldGuy.java

//older() - Returns true if this guy is older
//          Returns false if this guy is
//          younger or the same age
public boolean older(OldGuy him) {
  return age > him.age;
}

//younger() - Returns true if this guy is
//            younger
//            Returns false if this guy is
//            older or the same age
public boolean younger(OldGuy him) {
  return age < him.age;
}

Revising TestOldGuy()

public class TestOldGuy {
  public static void main(String[] args) {
    OldGuy me = new OldGuy(),
              him = new OldGuy();
    me.read();
    him.read();

    if (me.older(him))
      System.out.println("I\'m older.");
    else if (me.younger(him))
      System.out.println("I\'m younger.");
  }
}
Example: Complex Numbers

• Complex numbers are number of the type
  \[ w = x + iy \]
  where \( x \) and \( y \) are real and \( i \) is the square root of \(-1\).

• We can define the operations addition, subtraction and multiplication.

Complex Number Operations

• If our two complex numbers are \( u \) and \( v \):
  – If \( w = u + v \)
    • \( Re\ w = Re\ u + Re\ v \)
    • \( Im\ W = Im\ u + Im\ v \)
  – If \( w = u - v \)
    • \( Re\ w = Re\ u - Re\ v \)
    • \( Im\ W = Im\ u - Im\ v \)
  – If \( w = u \cdot v \)
    • \( Re\ w = Re\ u \cdot Re\ v - Im\ u \cdot Im\ v \)
    • \( Im\ w = Re\ u \cdot Im\ v + Im\ u \cdot Re\ v \)
import java.util.Scanner;

public class Complex {
  private double real, imag;

  // Accessors
  public double getReal() {
    return real;
  }

  public double getImag() {
    return imag;
  }

  // Mutators
  public void setReal(double newReal) {
    real = newReal;
  }

  public void setImag(double newImag) {
    imag = newImag;
  }

  // Write() - Write a complex value
  public void write() {
    System.out.print("(" + real + ", " + imag
            + ")");
  }
}
// Read() - Read in a Complex value
public void read() {
    Scanner keyb = new Scanner(System.in);

    System.out.println("Enter real\t?");
    real = keyb.nextInt();

    System.out.println("Imaginary\t?");
    imag = keyb.nextInt();
}

// Add() - Returns the sum of this value + v
public Complex add(Complex v) {
    Complex w = new Complex();
    w.real = real + v.real;
    w.imag = imag + v.imag;
    return w;
}

// Sub() - Returns the difference of
//          this value - v
public Complex sub(Complex v) {
    Complex w = new Complex();
    w.real = real - v.real;
    w.imag = imag - v.imag;
    return w;
}
// Mult() - Returns the product of this value times v
public Complex mult(Complex v) {
    Complex w = new Complex();
    w.real = real * v.real - imag * v.imag;
    w.imag = real * v.imag + imag * v.real;
    return w;
}
}

---

TestComplex.java

//TestComplex - Demonstrate the complex class
class TestComplex {
    public static void main(String[] args) {
        Complex u = new Complex(),
        v = new Complex(),
        w = new Complex();

        u.read();
        v.read();
        w = u.add(v);
        w.write();
        System.out.println();
Constructors

- Sometimes we need an object to have some initial values set when we define it. This can be done implicitly by writing a `constructor`.
- Constructors are called automatically when the program enters the function where the object is declared.
- Constructors share a name with the class and have no result type, not even void.
Default Constructors

• If an object is declared without any parameters, the default constructor is called.
• A default constructor has no parameters.

Initializing Constructors

• Initializing constructor initialize some or all of the values within the object.
• To use an initializing constructor, an object must be declared including (in parentheses) the initial values:
  
  ```java
  MyClass
  myObject = new MyClass(2, "name");
  ```
Adding Constructors to Complex.java

// Complex() - A Default Constructor
public Complex() {
    real = imag = 0.0;
}

// Complex() - An Initializing Constructor
public Complex(double a, double b) {
    real = a;
    imag = b;
}

// Complex() - A Conversion Constructor
Complex(int a, int b) {
    real = (double) a;
    imag = (double) b;
}

public static void main(String[] args) {
    Complex u = new Complex(1, 1),
            v = new Complex(),
            w = new Complex();

    v.read();
    w = u.add(v);
    w.write();
    System.out.println();
    w = u.sub(v);
    w.write();
    System.out.println();
    w = u.mult(v);
    w.write();
    System.out.println();
}