Intro to Java III: Objects

CS 1025 Computer Science Fundamentals I

Stephen M. Watt
University of Western Ontario

Our Previous Program

```
class Primes {
  public static boolean[] makeSieve(int n) {
      boolean[] marks = new boolean[n];
      for (int i = 0; i < n; i++) marks[i] = true;
      return marks;
  }
  public static void doCancel(boolean[] marks, int n) {
      if (! marks[n]) return; // ! means "not"
      for (int k = 2*n; k < marks.length; k += n) marks[k]=false;
  }
  public static void printPrimes(boolean[] marks) {
       for (int i = 2; i < marks.length; i++)
              if (marks[i]) System.out.print(" " + i);
  }
  public static void main(String[] args) {
       boolean[] sieve = makeSieve(100);
       for (int i = 2; i < sieve.length; i++) doCancel(sieve, i);
       System.out.print("Primes:");
       printPrimes(sieve);
       System.out.println(".");
  }
}
```

Classes as Collections of Related Things

- A class may declare variables as well as functions.
  The variables may be used by the functions
  if the word “static” is left off their declarations, *e.g.*

```
class Barn {
    int numChickens = 0;
    int numCows     = 0;

    public void addChickens(int n) { numChickens += n; }
    public void addCows(int n)     { numCows += n; }

    public int  numFeet() {
          return 4*numCows + 2*numChickens;
    }
    public int numEyes() {
          return 2*(numCows + numChickens);
    }
}
```

Using the Class -- Objects

- Notice that the **`Barn`** class did not have a **`main.`**
- We must therefore use it from another class that does have one.
- To do this we declare a variable of type **`Barn`** and use **`new`** to make one, e.g.

```
Barn myBarn = new Barn();
```

- The functions are called using "dot" notation:

```
myBarn.addChickens(4);
myBarn.addCows(3);

int nEyes = myBarn.numEyes();
System.out.println("Number of eyes = ", nEyes);
```

- **`myBarn`** is then said to be an *object* of type **`Barn`**.
- Its functions are called *methods.*

Multiple Objects

- There can be several *independent* objects with the same type in the program.

```
Barn myBarn    = new Barn();
Barn yourBarn  = new Barn();
Barn fredsBarn = new Barn();
```

- Several variables may refer to the *same* object:

```
Barn bigRedBarn = yourBarn;
```

No matter which name is used, the same object is affected.

```
bigRedBarn.addChickens(3); // affects yourBarn
```

Abstraction

- Objects allow you to provide programs without revealing how the data is represented.

- For example, you can represent a complex number in either polar (r, theta) or Cartesian (x, y) form

  and hide this from the user!

- This is an important idea -- it lets you change your mind.

  You can change the class and the programs that are using it still keep working.

- This is called “information hiding” or “data abstraction.”

Constructors

- Sometimes you want to initialize some of the variables differently for each object you construct.
- For this “constructors” are used.
- These are special methods with the same name as the class.

```
class Complex {
    private double _x, _y;
    public Complex(double x, double y) {_x = x; _y = y; }

    public double x() { return _x; }
    public double y() { return _y; }
    public double r() {
           return Math.sqrt(_x*_x + _y*_y);
    }
    public double theta() {
           return Math.atan2(_y, _x);
    }
}
```

Public vs Private

- Class variables are sometimes called “fields.”
- Constructors, fields and methods may be declared either **public** or **private.**
- Only **public** items can be accessed from outside the class using the dot notation.