Classes: School is in Session

Aneesh Karve
Abstraction

- simplified, *high level* view
- captures essential properties
Encapsulation

- public interface encapsulates (hides) the private implementation
- allows non-experts to use expert implementations as a “black box”
Abstraction in automobiles
Abstraction in software
Object-oriented Programming (OOP)

• way of thinking about complex problems
• provides abstraction and therefore power
• build complex programs out of black box components
Advantages of OOP

• makes programs easier to think about
  – fewer bugs
  – easier to maintain

• can create custom classes (types)
Designing Classes

- first think of behavior (public interface); worry about implementation details later
Access modifier

- keyword which determines availability of class, field, or method
- examples
  - public
  - private
private enforces encapsulation

```java
public class Capsule{
    private String secret = "rosebud";

    public void changeSecret(String code){
        secret = code;
    }
}
```

OK to access private fields from within class

```java
public class Spy{
    public static void main(String[] args){
        Capsule c = new Capsule();
        String stolen = c.secret;
    }
}
```

Can't access from outside
Class syntax

public class RockBand{
    fields (also called "data members" or "instance variables")
    private String name = "Sound Garden";
    ...
    constructors
    public RockBand(String name){...}
    ...
    methods
    public String getName(){...}
    ...
}
Fields

- each object has the instance fields specified by its class
- generally, fields should be private
Field declaration

- inside of class definition

```java
private int code = 33;
```
Constructors

- create new objects
- initialize instance fields
- have same name as class
Constructor example

```java
public class IPod{
    //field
    private double cost;
    //constructor
    public IPod(double price){
        //initialize cost field
        cost = price;
    }
}
```
Method anatomy

```
public int addThreeNumbers(int num1, int num2, int num3){...}
```

Method Body

• where the work gets done
• consists of *statements*
• contains zero or more return statements
return

- halts method execution
- may return a value
Return type

• type returned by method

```java
public int someMethod()
{
    int x = doABunchOfCalculations();
    return x;
}
```

```java
public void anotherMethod()
{
    // code goes here
    return; // methods that don't return a value have return type void
}
```
this

- object reference to implicit parameter

```java
public class IPod {
    //field
    private double cost;
    //constructors ...
    //method
    public void setCost(double cost) {
        this.cost = cost;
    }
}
```
Javadoc

- tool for generating HTML documentation from source code
- use liberally (before each field, method, class, constructor)
- documentation is essential to intelligible, reusable code
Javadoc in action

[Diagram showing the process of generating HTML documentation from Java source code using Javadoc]
Javadoc syntax

/** This method is rad. Here's why:
 * <B>cookie monster is the man!</B>.
 * I think he should have his own show.
 * @param yum the cookie to eat
 * @param c number of yums to eat
 * @return number of crumbs flying from mouth
 */

public int eatCookies(String yum, int c){...}
### Categories of Variables

<table>
<thead>
<tr>
<th></th>
<th>where declared?</th>
<th>where initialized?</th>
<th>lifetime?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>instance</strong></td>
<td>class</td>
<td>constructor</td>
<td>object</td>
</tr>
<tr>
<td><strong>parameter</strong></td>
<td>method prototype</td>
<td>method call</td>
<td>method execution</td>
</tr>
<tr>
<td><strong>local</strong></td>
<td>block*</td>
<td>block (or else error)</td>
<td>block</td>
</tr>
</tbody>
</table>

* a block of code is enclosed by brackets: `{}`
Local vs. Parameter variables

• **similarity: defined and initialized in methods**
• **difference: how initialized**
  – parameter variables are initialized by calling the method
  – local variables initialized using assignment operator in method body
Garbage collector

- reclaims memory from unreachable objects
- garbage not always collected immediately