CS 211

Classes, Objects

Topics

- Classes and Objects – concept
- Classes and Objects – usage
- Brief Method introduction
- Constructor Methods

Classes and Objects

Classes and Objects

What is a class? (What do we place in a class?)

What is an object? (where do we create them?)

How do we create a class?

How do we create an object?

How do we use an object?

A class is a Type

A **Class** defines a new type, from which we can create values (**objects**). The class definition specifies:

what **state** is in each value (aggregating values)

what **behaviors** the values can exhibit (methods)

Classes

- many values are of type `int`
- we can create many values of our class type.

A class is a "blueprint" for making multiple values (objects) that have the same structure/methods.

- class types are **reference types** → we get references pointing to the objects stored in memory

An object is a value

An **object** is called an **instance** of a class
(a specific value of that reference type).

From one class definition we can make many unique objects, each with their own state (sub-values).

Objects are just like other values:

- we can create variables to hold objects
- create arrays to hold many same-type references to objects

Class Components

The class's name is the newly defined type's identifier.

State: declaring a variable directly in a class represents an **instance variable**: each object will have its own value for each instance variable.

Behaviors: a **method** defined directly in a class can be called on any object of that class, using that object's instance variables.

Class Example – IceTray class

State (instance variables)

- `count, capacity, ready`
- each IceTray has their own instance variables

Behaviors: (methods)

- `fill, freeze, take, isReady`
- every IceTray can use these methods on their own state

```
public class IceTray {
  int count;
  int capacity;
  boolean ready;

  public void fill(){
    if (count!=capacity){ ready=false; }
    count = capacity;
  }
  public void freeze(){
    ready = true;
  }
  public int take(int n){
    if (!ready){return 0;}
    int ans = n<=count? n : count;
    count -= ans;
    return ans;
  }
  public boolean isReady() { return ready;}
}
```

Class syntax

A class definition is: any modifiers, the class keyword, followed by the class's identifier, followed by curly braces. In the curly braces two different sorts of definitions are allowed:

```
modifiers class identifier {
        fieldDefinitions

        methodDefinitions

}
```

field definitions: Declares fields (variables) associated only with this class.

method definitions: Declares methods associated only with this class.

simplifications: we are ignoring unlearned syntax for now.

Class Syntax Notes

Modifiers: modifiers (like **public**, **private**, **protected**, **final**) can make substantial changes to the meaning of the variables and methods of a class. We will study these in more details throughout the semester.

Quick Tour:

→ public, private, protected: controls who can access it
→ static: controls whether it's a single shared thing or tied to a specific object

→ final: disallows further changes
→ abstract: thing can be extended, but not directly used

→ synchronized, volatile: used in threading

Ordering: Java allows any order of fields/methods (all called "members").
→ convention: fields first, methods second. (constructor methods first).

Object Creation

To create an object, we call the class's constructor method.

Syntax: the `new` keyword, the class type (the class's name), and an argument list in parentheses for the constructor method.

Example: `new IceTray()`

The default constructor has no parameters, but we can create our own constructors that do much more (discussed later).

Semantics: Java uses the constructor method and makes space in memory for another object (space for all instance variables).

Initial values are set up according to the constructor method's code. A ***reference*** to this spot in memory is the result of evaluating this constructor call (hence the name "reference type").

Object Example

`t` is a reference to an object having the type `IceTray` (it is an `IceTray` value)

we access/update an instance variable by:

`objExpr.instVarName`

ex: `t.ready`

we ask an object to call its method on itself by:

`objExpr.methodName(args)`

ex: `t.take(2)`

```
// create an object (call constructor)
IceTray t = new IceTray();

// manipulate it
t.ready = false;
t.capacity = 12;
t.fill();

// interact with it: call methods, print
System.out.println("#:",t.count);
System.out.println("got:",t.take(3));
t.freeze();
```

Object Uniqueness

Multiple objects of one type can be created that are distinct.

Each occupies a separate spot in memory.

```
IceTray t1 = new IceTray();
IceTray t2 = new IceTray();

t1.capacity = 10;
t2.capacity = 36;

System.out.println(t1.capacity);
System.out.println(t2.capacity);


t1.fill();
t1.freeze();

System.out.println(t1.isReady());
System.out.println(t2.isReady());
```

Objects are values

- Objects are just values of a particular type

"Every expression has a type": class-definitions are types too!

- Object-yielding expressions result in a reference to an object
  - constructor calls (actually, the **new** keyword)
  - methods with a return type that is a class-name

Practice Problems

Create a class to represent a **Coordinate**
(an x and y value representing two dimensions)

- What instance variables should this class have?

Create a class to represent a **Square**

- What instance variables should this class have?

Pytania Poll

- **Classes and Objects**

Quick Note on Using Classes

We put one class definition in each file, and give the file the class's name: ClassName.java.

- For files in the same directory, we can just use another class by name:
  TestIceTray.java (in same directory as IceTray.java)
- we will look at packages later, as a way to organize all these class files.

```
class TestIceTray{
    public static void main (String[]args){
        //We can just use the IceTray class directly.
        IceTray t1 = new IceTray();
        t1.capacity = 12;
    }
}
```

Methods (Brief Review/Introduction)

method: named block of code that can be called. It is like a function that is associated with a specific object or class.

Our first view of methods:

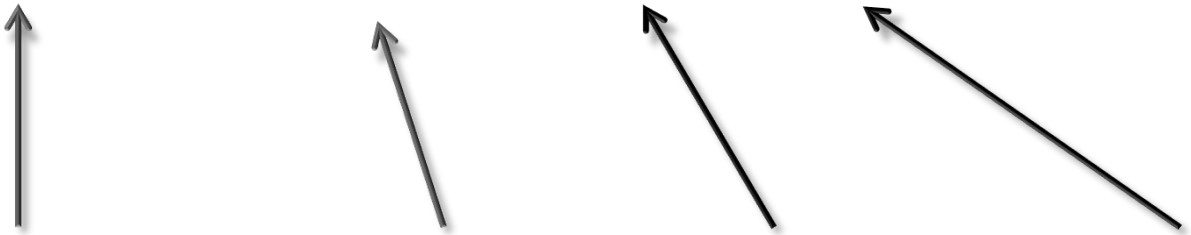
- defined in a class.
- Requires an object of that class type
- "calling" a method: asking the object to run the code for us
- object performs the call using its own instance variables.

→ *this is almost the whole story, but not quite!*

Methods (Brief Introduction)

Method Signature: all the modifiers, type information (return type and parameters), and the name. Provides all details needed for interacting with/identifying the method.

```
public String getItem(int i){...}
```



modifiers (public,private, static, etc)

return type

method name

parameter list

Methods (Brief Introduction)

When we define a method we must:

- ***give the return type***: what type of value will be returned? If no value should be returned, return type is listed as `void`.

- ***define the listing of parameters:*** types and names for values that are supplied (as an 'argument list') at each method call.

- ***write the method body*** (block of code) using parameters to perform some task and return a value of the indicated return type.

Method Example - take

Modifier: **`public`**.
(visible outside an object)

Return type: **`int`**. the method must return a value, and it must be an int.

Name: **`take`**.

Parameter list: **`(int n)`**.
variable declarations (without instantiations) that are available only in this method call

Method Body: uses/changes the object's state and returns an int.

```
class IceTray{
  int count, capacity;
  boolean ready;
  …
  public int take(int n){
    if (!ready) { return 0;}
    int ans = (n<=count) ? n : count;
    count -= ans;
    return ans;
  }
 …
 }
```

Practice Problems

Coordinate Class:

Add a method that calculates the distance from the origin. (Use Pythagoras' Theorem).

- use Math.sqrt and Math.pow. (java.lang.Math is always available)

Square Class:

- Add methods to calculate the area and the perimeter.

Constructor Method

constructor method: special method that is used to create a new object-value of the class. It implicitly returns a reference to this new object.

Return Type: *not specified*, because the constructor always returns a reference to an object of the class type.

Method Name: always identical to the class name.

Parameters: Just like regular methods (can have zero or more). Often, parameters are (nearly) one-to-one matches of instance variables.

Body: chance to set initial values for instance variables, perform consistency checks, do any other involved setup work (including calling other methods)

IceTray Example: Constructor

Parameter list: we chose one for each instance variable.

body: initialized all instance variables (based on params).

Point of No Return! A constructor doesn't need an explicit return statement to return the object, because the **new** keyword is what returns the new object, not the constructor.

```
class IceTray {    ...
  public IceTray (int count , int capacity, boolean ready){
    this.count = count;
    this.capacity = capacity;
    this.ready = ready;
  }
}
```

What is this madness?

Code inside a class can refer to itself via the **this** keyword.

- it only makes sense with non-static methods (we have an object)
- this is the same as Python's use of **self**.

```
public class Triplet {
    public int x,y,z;
    public Triplet(int x, int y, int z){
        this.x = x;
        this.y = y;
        this.z = z;
    }
}
```

Practice Problems

Coordinate Class:

Add a constructor method. What parameters should it have?

Square Class:

- Add a constructor method. What parameters should it have?

Method Overloading (Quick View)

We can have **multiple methods with the same name** in one class! They are distinct methods with no actual relations other than the name – this is just a mnemonic convenience!

To coexist, **their method signatures must have unique sequences of parameter types).**

- parameter names are ignored – arguments are nameless.
- return type is ignored – not explicit via method call, so can't help select correct method.

Constructors are methods too – we can write multiple constructors for one class, as long as they have different signatures.

- This is a valuable opportunity (something Python disallows).

Method Overloading - Examples

Example: these methods may all be in one class:

```
1. public int foo (int a, int b){…}
2. public int foo (char a, int b){…}
3. public int foo (int b, char a){…}
4. public int bar (int a, int b){…}
5. public String foo (int a, int b, int c){…}
```

The following could not be added to the class:

```
public String foo (int a, int b) {…}          // return type irrelevant
public int foo (int other, int names){…}      // param names irrelevant
private int foo (int a, int b){…}             // modifier irrelevant
```

Practice Problems

Method Overloading:

Add another constructor method to the Coordinate, Square, and IceTray classes.

→ How will it be distinguished from the other constructor method(s)?

Default Constructors

Default Constructor: Java provides a default constructor definition when none is present in a class: there are no parameters, and all instance variables get default values: primitive types get `0`, `0.0`, `false`; reference types (class types) get the `null` value.

null is a special value: it represents a value of any reference type but has no actual reference value (no instance variables, no methods). Attempting to use null like an actual object is a very common run-time error. (a `NullPointerException`)

Pytania Poll

- **Methods**