CS206

I/O Methods

Files/Exceptions

Inheritance
In this view, lines indicate shared traits. This is the “inheritance” in OO.
public class Petv1 extends Object {
    private String iD;
    private String name;
    public String sound() {
        return "silence";
    }
    public String getId() {
        return iD;
    }
    public String getName() {
        return name;
    }
}

public class Pet {
    protected String iD;
    protected String name;
    public String sound() {
        return "silence";
    }
    public String getId() {
        return iD;
    }
    public String getName() {
        return name;
    }
    public boolean equals(Pet p) {
        return iD.equals(p.getId());
    }
}
public class Cat extends Pet {
    private String breed;
    private double hairLength;
    public Cat(String name, String id, String breed) {
        this.name = name;
        this.iD = id;
        this.breed = breed;
    }
    @Override
    public String sound() {
        return "meow";
    }
    @Override
    public String toString() {
        return "My name is " + name + " breed " + breed + " and I say " + sound();
    }
    public static void main(String[] args) {
        System.out.println(new Cat("calypso", "112234", "siberian"));
    }
}
Dog Classes

public class Dog extends Pet{
    protected String group;
    protected double hairLength;
    protected boolean doubleCoat;
    @Override
    public String sound() {
        return "arf";
    }
    @Override
    public String toString() {
        return sound();
    }
    public static void main(String[] args) {
        System.out.println(new Dog());
    }
}

public class WorkingDog extends Dog {
    protected String breed;
    protected String task;
    @Override
    public String toString() {
        return super.toString() + " work " + task;
    }
    @Override
    public String sound() {
        return "woof";
    }
}

}
Casting, Classes and Inheritance

- Suppose: SPCA pet shelter
- Desire: A program that tracks all animals at shelter
- Approach
  - Use single array to hold all Pets
  - But deal with dogs cats separately later

```java
public class Shelter {
    Pet[] animals = new Pet[100];
    int animalCount=0;
    public void addAnimal(Pet animal) {
        animals[animalCount++]=animal;
    }
    public Pet getAnimal(int location) {
        return animals[location];
    }
    public static void main(String[] args) {
        Shelter shelter = new Shelter();
        shelter.addAnimal(new Dog());
        shelter.addAnimal(new Cat());
        Pet aa = shelter.getAnimal(1);
        System.out.println(aa);
        if (aa instanceof Cat) {
            Cat c = (Cat)aa;
            System.out.println(c.toString());
        }
    }
}
```

Unlike base types, objects remember what they were
casting of class “up” to Pet
cast “down” to Cat
Class Organization and main

- Generally, main should not do the work.
  - It should be short!
  - Do not put a significant function of a class into main.
- Main can be large, if its only purpose is examples of use of class.
- Parameterize whenever possible

```java
public class Fibonacci {
    public void doIt(int limit) {
        int n_2=1;
        int n_1=1;
        for (int i=1; i<limit; i++) {
            int nI = n_2 + n_1;
            System.out.println(n_1 + " "
                              + n_2 + " " + nI + " " + ((double)nI / n_2));
            n_1 = n_2;
            n_2 = nI;
        }
    }
    public static void main(String[] args) {
        new Fibonacci().doIt(20);
    }
}
```
Exceptions

• Unexpected events during execution
  ▫ unavailable resource
  ▫ unexpected input
  ▫ logical error

• In Java, exceptions are objects

• 2 options with an Exception
  • “Throw” it
    • this says that the exception must be handled elsewhere
  • “Catch” it.
    • handle the problem here and now
Catching Exceptions

- Exception handling
- \texttt{try-catch}
- An exception is caught by having control transfer to the matching \texttt{catch} block
- If no exception occurs, all \texttt{catch} blocks are ignored

\begin{verbatim}
try {
    guardedBody
} catch (exceptionType_1 variable_1) {
    remedyBody_1
} catch (exceptionType_2 variable_2) {
    remedyBody_2
} ...
  ...
\end{verbatim}
Throwing Exceptions

• An exception is thrown
  □ implicitly by the JVM because of errors
  □ explicitly by code
  □ If your code throws an exception it must catch that exception somewhere else

• Method signature – throws

  public static int parseInt(String s) throws NumberFormatException
public class ExceptThrower {
    public int divv(int numer, int denom) {
        try {
            return numer / denom;
        } catch (ArithmeticException e) {
            System.err.println("Caught in Func "+ e);
        }
        return 0;
    }
    public int divvTh(int numer, int denom) throws ArithmeticException{
        return numer / denom;
    }
    public static void main(String[] args) {
        ExceptThrower except = new ExceptThrower();
        except.divv(2, 0);
        try {
            except.divvTh(4, 0);
        } catch (ArithmeticException ae) {
            System.err.println("Caught in Main "+ ae);
        }
    }
}
Java’s Exception Hierarchy

- Exception
  - IOException
  - SQLException
  - AWTException
  - EOFException
  - FileNotFoundException
  - InterruptedIOException
  - FileNotFoundException
  - RuntimeException
    - ArithmeticException
    - NullPointerException
    - ClassCastException
    - IndexOutOfBoundsException
      - ArrayIndexOutOfBoundsException
      - StringIndexOutOfBoundsException
  - Error
    - StackOverflowError
    - VirtualMachineError
    - OutOfMemoryError
    - AssertionError
    - ExceptionInInitializerError
    - IOException
    - AWTError

RuntimeException & its sub-classes and Error & its sub-classes are Unchecked Exception; All other exceptions are Checked Exception
Reading the Keyboard

• System.in is, by default, set to receive keyboard input

• Use Scanner to read from keyboard
  • Do NOT use scanner otherwise

```java
import java.util.InputMismatchException;
import java.util.Scanner;
public class CatKeyboard {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.print("Enter name: ");
            String name = scanner.next();
            System.out.print("Enter Id:");
            int iid = scanner.nextInt();
            System.out.print("Enter breed: ");
            String breed = scanner.next();
            Cat ccat = new Cat(name, (""+iid), breed);
            System.out.println(ccat);
        } catch (InputMismatchException ime) {
            System.out.println(ime);
        }
    }
}
```
Handling Exceptions — alternate cat try-catch

```java
public Cat() throws InputMismatchException {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter name: ");
    name = scanner.next();
    System.out.print("Enter Id:");
    iD = "" + scanner.nextInt();
    System.out.print("Enter breed: ");
    breed = scanner.next();
}

public Cat getCat() {
    try {
        Cat c = new Cat();
        return c;
    } catch (InputMismatchException ime) {
        System.err.println(ime);
    }
    return null;
}
```

Exceptions should be handled as soon as possible.

try-catch should enclose as little code as possible
public void readOneLineTC()
{
    BufferedReader br;
    try {
        br = new BufferedReader(new FileReader(fileName));
        br.readLine();
    }
    catch (FileNotFoundException fnf) {
        System.err.println("No file " + e);
    }
    catch (IOException e) {
        System.err.println("Reading " + e);
    }
    finally {
        if (br!=null) {
            try {
                br.close();
            }
            catch (IOException ioe) {
                System.err.println("Close" + ioe);
            }
        }
    }
}

public void readOneLineTCR()
{
    try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
        br.readLine();
        // close unnecessary in this formulation
    } catch (FileNotFoundException e) {
        System.err.println("Open " + e);
    } catch (IOException e) {
        System.err.println("Reading " + e);
    }
}

finally == code that WILL be executed. Optional part of try-catch
Close can throw an exception so it too must be caught. Sigh
Software Design Goals

• Robustness
  ▫ software capable of error handling and recovery
  ▫ programs should never crash
    ▫ ending abruptly is not crashing

• Adaptability
  ▫ software able to evolve over time and changing conditions
    (without huge rewrites)

• Reusability
  ▫ same code is usable as component of different systems in various applications
  ▫ The story of Mel — [https://www.cs.utah.edu/~elb/folklore/mel.html](https://www.cs.utah.edu/~elb/folklore/mel.html)
OOP Design Principles

• Modularity
  • programs should be composed of “modules” each of which do their own thing
    • each module is separately testable
  • Large programs are built by assembling modules
  • Objects (Classes) are modules

• Abstraction
  • Get to the core — non-removable essence of a thing
  • Most pencils are yellow, but yellowness does not required

• Encapsulation
  • Nothing outside a class should know about how the class works.
    • For instance, does the Object class have any instance variables. (Of what type?)
  • Allows programmer to totally change internals without external effect
OOP Design

• Responsibilities/Independence: divide the work into different classes, each with a different responsibility and are as independent as possible

• Behaviors: define the behaviors for each class carefully and precisely, so that the consequences of each action performed by a class will be well understood by other classes that interact with it.
Class Definition

- Primary means for abstraction in OOP
- Class determines
  - the way state information is stored – via instance variables
  - a set of behaviors – via methods
- Classes encapsulate
  - private instance variables
  - public accessor methods (getters)
Constructors

• Constructors are never inherited
• A class may invoke the constructor of the class it extends via a call to `super` with the appropriate parameters
  • e.g. `super()`
  • `super` must be in the first line of constructor
  • If no explicit call to `super`, then an implicit call to the zero-parameter `super` will be made
• A class may invoke other constructors of their own class using `this()`
  • `this` must be first
  • Cannot explicitly use both `super` and `this` in single constructor
• See FileOpen.java for example
Lab

Do Feb 16 lab from the class website.

You will use most of this in HW 1 which just adds comments and submits using the homework submission system.