CS206

I/O Methods

Files/Exceptions

Inheritance
Strings

- Strings - "a", "abc" — double quotes
- Characters - ‘a’ — single quotes
- Declaring String objects
  ```java
  String name;
  String name = new String();
  ```

- Declaring String objects with initialization
  ```java
  String name1 = "Fred";
  String name2 = new String("Fred");
  ```

There are subtle differences between these two declarations.
.equals: Object Equality

- Use == only when comparing base types
  - int, float, ...
- Otherwise use .equals
Wrapper Types

• Most data structures and algorithms in Java’s libraries only work with object types (not base types).

• To get around this obstacle, Java defines a wrapper class for each base type.

• Implicitly converting between base types and their wrapper types is known as automatic boxing and unboxing.
public class Wrapper
{
    public void w1(Integer ii) {
        System.out.println(ii);
        int i3 = ii; // auto unboxing
        System.out.println(i3*i3);
        System.out.println(i3*ii); // auto unboxing
    }
    public static void main(String[] args) {
        Wrapper w = new Wrapper();
        w.w1(5); // autoboxing
    }
}
What you should know/review

- variables
- expressions
- operators
- methods
  - parameters
  - return value
- conditionals
- for/while loops
- class design and object construction
  - instance variables
  - constructor
  - getters/setters
  - class methods
  - new
- arrays
- arrays of objects
- String
Start of the Java class hierarchy

http://web.deu.edu.tr/doc/oreily/java/langref/ch10_js.htm
Java Object Methods

- public boolean equals(Object ob)
- public String toString()
- public Class getClass()

- protected Object clone()
- protected void finalize()
- public int hashCode()
- public void notify()
- public void notifyAll()
- public void wait()
- public void wait(long l)
- public void wait(long l, int ii)
Casting, Classes and Inheritance

• Suppose: SPCA shelter for only dogs and cats
• Desire: A program that tracks all animals at shelter
• Approach
  • Create 3 classes, Dog and Cat that extend (inherit from) from Animal.
  • Use single array to hold all animals
  • But deal with dogs cats separately later

```java
public class Animal {
    // Animal class definition
}

public class Dog extends Animal {
    // Dog class definition
}

public class Cat extends Animal {
    // Cat class definition
}

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

• An exception is caught by having control transfer to the matching catch block

• If no exception occurs, all catch blocks are ignored

```java
try {
    guardedBody
}
catch (exceptionType_1 variable_1) {
    remedyBody_1
}
catch (exceptionType_2 variable_2) {
    remedyBody_2
}
...
...
```
Throwing Exceptions

• An exception is thrown
  ▫ implicitly by the JVM because of errors
  ▫ explicitly by code

• Exceptions are objects
  ▫ throw an existing/predefined one
  ▫ make a new one

• Method signature – `throws`

  ```java
  public static int parseInt(String s)
  throws NumberFormatException
  ```
Java’s Exception Hierarchy

- **Object**
  - **Throwable**
    - **Exception**
      - **RuntimeException**
      - **IOException**
      - **SQLException**
      - **AWTException**
    - **Error**
      - **StackOverflowError**
      - **VirtualMachineError**
      - **AssertionError**
      - **OutOfMemoryError**
      - **ExceptionInInitializerError**
      - **IOException**
      - **AWTError**

- **Unchecked Exception** (RuntimeException & its sub-classes and Error & its sub-classes are Unchecked Exception; All other exceptions are Checked Exception)

- **Checked Exception** (RuntimeException & its sub-classes and Error & its sub-classes are Unchecked Exception; All other exceptions are Checked Exception)

CS206 13 Lec02
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
class Student {
    String name;
    int age;

    public Student(String n, int a) {
        name = n;
        age = a;
    }

    public String toString() {
        StringBuilder sb =
            new StringBuilder("Details..............");
        sb.append("\nName: ").append(this.name);
        sb.append("\nAge: ").append(age);
        return sb.toString();
    }

    public Student() throws IOException, InputMismatchException {
        this(); // call the default constructor to be sure that the variables are initialized
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.print("Enter student name: ");
            name = scanner.nextLine();
            System.out.print("Enter Age: ");
            age = scanner.nextInt();
        } finally {
            // cleanup
        }
    }
}
```
Handling Exceptions

**try-catch**

```java
public static void main2(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String name;
    int age;
    System.out.print("Enter student name: ");
    name = scanner.nextLine();
    try {
        System.out.print("Enter Age: ");
        age = scanner.nextInt();
    } catch (InputMismatchException e) {
        System.err.println("problem " + e);
        return;
    }
    Student student = new Student(name, age);
    System.out.println("\n" + student.toString());
}
```

Exceptions should be handled as soon as possible.

try-catch should enclose as little code as possible.

main2 — looks like main but will not be executed.
public Student(InputStream inStream) throws IOException, InputMismatchException {
    this(); // call the default constructor to be sure that the variables are initialized
    Scanner scanner = new Scanner(inStream);
    System.out.print("Enter student name: ");
    name = scanner.nextLine();
    System.out.print("Enter Age: ");
    age = scanner.nextInt();
}

public static void main(String[] args) {
    try {
        Student student = new Student(System.in);
        System.out.println("\n" + student);
    } catch (IOException ioe) {
        System.err.println("problem " + ioe);
    } catch (InputMismatchException ime) {
        System.err.println("problem2 " + ime.toString());
    }
}
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
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