CSCB07

Albion Fung ~(._.)~
github.com/conanap - CSCB07F16Seminar and CSCB07F16F
albion@amacss.org

Software Version Control (SVN)

- Provides a way for different people to work together
- Organize different versions
- Track back to older versions
- See changes

- svn add
- svn checkout
- svn commit
- svn update
- svn commit
- svn delete
- svn status
- svn revert
- svn log -r 1:HEAD

svn checkout link

svn status

- ?: untracked
- A: Added but not committed yet
- M: modified and not committed yet
- C: conflict!

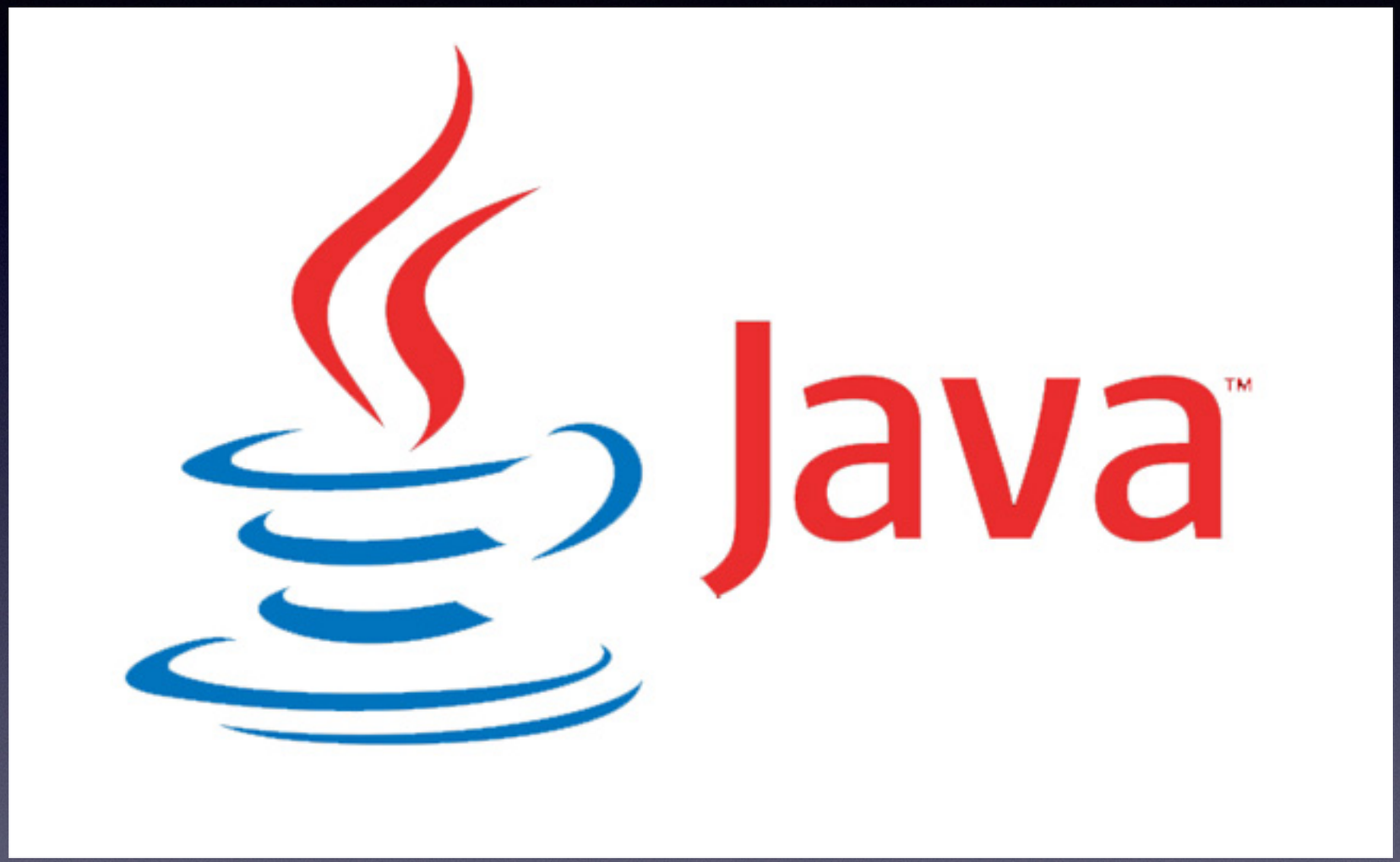
conflicts

- I don’t get it either.
- jk I figured it out

Conflicts

- On initial conflict, svn shows you the differences
- Either postpone it to solve later or do it now

- svn delete: removes a file
- svn log: shows you the commit logs
- svn revert: revert changes to most recent commit
- svn update -r# filename: “update” file name to revision #


Java™

Primitive types

- int (and long, short, etc)
- double (and float)
- boolean
- char

String is NOT a primitive type

- String a = “hi”; // immutable object
- String a = new String(“hi”); // object

Wrapping

- Basically an object version of primitives
- Why? I don’t fucking know
- Have some useful methods… that I never use
- Capitalize first letter and make it full word
- int -> Integer
- double -> Double

IMO totally useless but hey

Commenting

- // blah blah
- /*black sheep*/
- /**i forgot the next line*/ //JavaDocs

Java docs

- use /** */
- @param paramname paramdesc
- @author name
- @return desc

```
/**
 * A method that returns the number of pieces of chocolates I have yet to eat but I will.
 * @author Albion
 * @param name Name of the person
 * @param choc Type of chocolate
 * @return number of pieces
```

Java methods, inheritance and objects

Key diffs of Inheritance

- It’s basically python
- Except it only inherits 1
- Inherits only public and protected
- But yeah basically python

Constructors

- no method name - the class name is the constructor name

```
public class Aye {

  // declare vars here; NOT INIT

  public Aye(params) { }; // constructor; init vars here

  // don't return anything

   // can be private!

   }

}

Aye poop = new Aye(params here);
```

Super and this

- this refers to the current object
  - python equiv is self - but you don’t need to declare this in your args

- super = whatever of the supertypes!
  - super(); // call supertype constructor
  - super.a; // access supertype var a if public / protected

Factory Methods

- Make constructor private
- Provide multiple static methods to make new objects
- have specific and diff parameters to help differentiate
- clarifies purpose of each factory

```
private Circle() {…};

public static Circle MakeWithDegree(int radius,
double degrees) {…};

public static Circle MakeWithRadian(int radius,
double radian) {…};
```

Overloading vs Overriding

- Overriding = same return, signature
- Overloading = same signature diff input

Casting

- Nugget extends Food
- Food d = new Nugget(); // a okay
- Nugget s = new Food(); // not okay, Food is not a nugget (thank god)
- Nugget so = (Nugget);
- Food joke = new Drink(); // not okay, her jokes are shit
- Nugget son = new Nugget();
- Nugget what = (Food)son; // not okay, the food was not a nugget to begin with

Abstract

- Tfw you just need a mold
- abstract prefix
- If inherited, must have a physical implementation in the child
- Abstract classes may have some implemented methods

Interfaces

- Like abstract class, except no implementation at all
- private, public and protected inheritance rules

Inheritance vs Interface

- Inheritance: A baby duck it IS a duck.
- Interface: A robot duck BEHAVES like a duck but is NOT a duck

Interface vs Abstract

- Abstract if same implementation
- BUT can implement multiple interface
    - Don’t forget you can implement interface in abstracts

Generics

- Data<type, type,…>
- Basically to guarantee correct return type
- ArrayList<String> a = new ArrayList<String>();

Junit test

```
public class ??() {
    @Before
    (function here)
    @Test
    (test function here)
    @After // clean up
    (function here)
}
```

Exceptions

- Checked vs unchecked
- Inherit from Throwable

- assertTrue
- assertEquals
  - may need to write your own assertEquals
    - Reflexive => x.equals(x) == true
    - Symmetric => a.equals(b) == b.equals(a)
    - Transitive a.equals(b) && b.equals(c) = a.equals(c)

Polymorphism

- A type is the super type of many subtypes
- Best used when need to pass around a lot of diff types
- Reduce code clutter

Downcasting

- Casting a supertype to a subtype

```
class Animal { public void walk(), run() }

class Cat { public void purr() }

class Dog { public void bark() }

Animal lowkeyCat = new Cat();

Animal highkeyDog = new Dog();

lowkeyCat.purr();

((Cat) lowkeyCat).purr();

((Cat) highkeyDog).purr();
```

Static vs non static

- Static: bind at compile time
    - Cannot create an instance (effectively singleton)
- Dynamic: bind at run time
    - Requires an instance to use

ANY. QUESTIONS.?

Design Patterns

Publish Subscribe

- Subscribers must register
- Publisher notifies subscribed when something happens

Singleton Design

- Only one instance
- File system: can only have 1 FS!

Iterator

- For traversing data structures
- Can modify implementation of ADTs, access stay the same and universal
    - You don’t have to look up docs!

Dependency Injection

- Pass objs into constructor instead of creating inside
- Helps decouple code
- Allows better testing with mock objs

Nested classes

- Can declare a class in a class
- When only a class uses the nested class
- Can be static or non-static
- Can be public or private

Builder

- What happens when you have 500 optional vars in constructor?

Cry

Make all the possible combinations of constructors

- Only 1 constructor
- Pass in stuff you only need
- Requires nested class

```
class Nutritions {
   public static class Builder {
      // init all params here
      public Builder(mandatory args) {}
      public Builder (opt arg1) {
         opt1 = arg1; // decl'd above
         return this;
      } …
      public Nutritions build() {
         return new Nutritions(this);
      }
   }
   private Nutritions(Builder b) { // set stuff here }
}
```

```
Nutrition boop = new Nutrition.Builder(1,2)
                                .wow(“ikr”)
                                .sugar(true)
                                .build();
```

Things not covered but you should totally study

- UML & CRC Cards (!!important)
- Java Ant, Build file in XML
- Understanding of memory, stack and heap
- JVM, Java compilation process
- Agile vs waterfall