CSCB07S18F (Abbas)

Albion Fung
github.com/conanap

SVN

- svn co [addr]
- svn update
- svn commit -m “description”
- svn status
- svn add [file]
- svn delete [file]

Conflicts

Java and OO

Primitive Types

- Size in bits
- byte (8), short (16), int (32), long (64)
- float (32), double (64)
- boolean (1)
- char (16)

Array

- int[] intar = new int[10];
- notice keyword new
- Object!
- int[] intar = {1, 2, 3};

String

- Is object
- String str = “abc”; // immutable obj
- String str = new String({‘a’, ‘b’, ‘c’}); // obj

Wrapping

- Basically an obj version of primitives
- Why? I don’t know
- Have some useful methods that aren’t really ever used
- Capitalize first letter and make it full word
- int -> Integer
- double -> Double
- eg: Double dub = new Double(42.1);

IMO totally useless but hey

Commenting

- // sick
- /* sick with new lines */
- /**This function doesn’t do anything but if we remove it everything crashes*/ // javadocs

Java Docs

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

```
/**

* A method that returns the number of pieces I have yet
to eat

* @author Albion

* @param name Name of the person

* @param choc Type of chocolate

* @return number of pieces yet to have been eaten

*/
```

Memory in Java: Stack

- Stack is for temp vars
  - when var is out of scope, auto deallocate
- Faster on stack than heap
- Used for param parsing
- Stack overflow
- Stored in RAM

Memory in Java: Heap

- Also in RAM
- Anytime you use “new” keyword (ie, objs)
- Slower to alloc
- Must be removed manually
    - Note: Java does this for you when there are no more references to the obj

Scoping

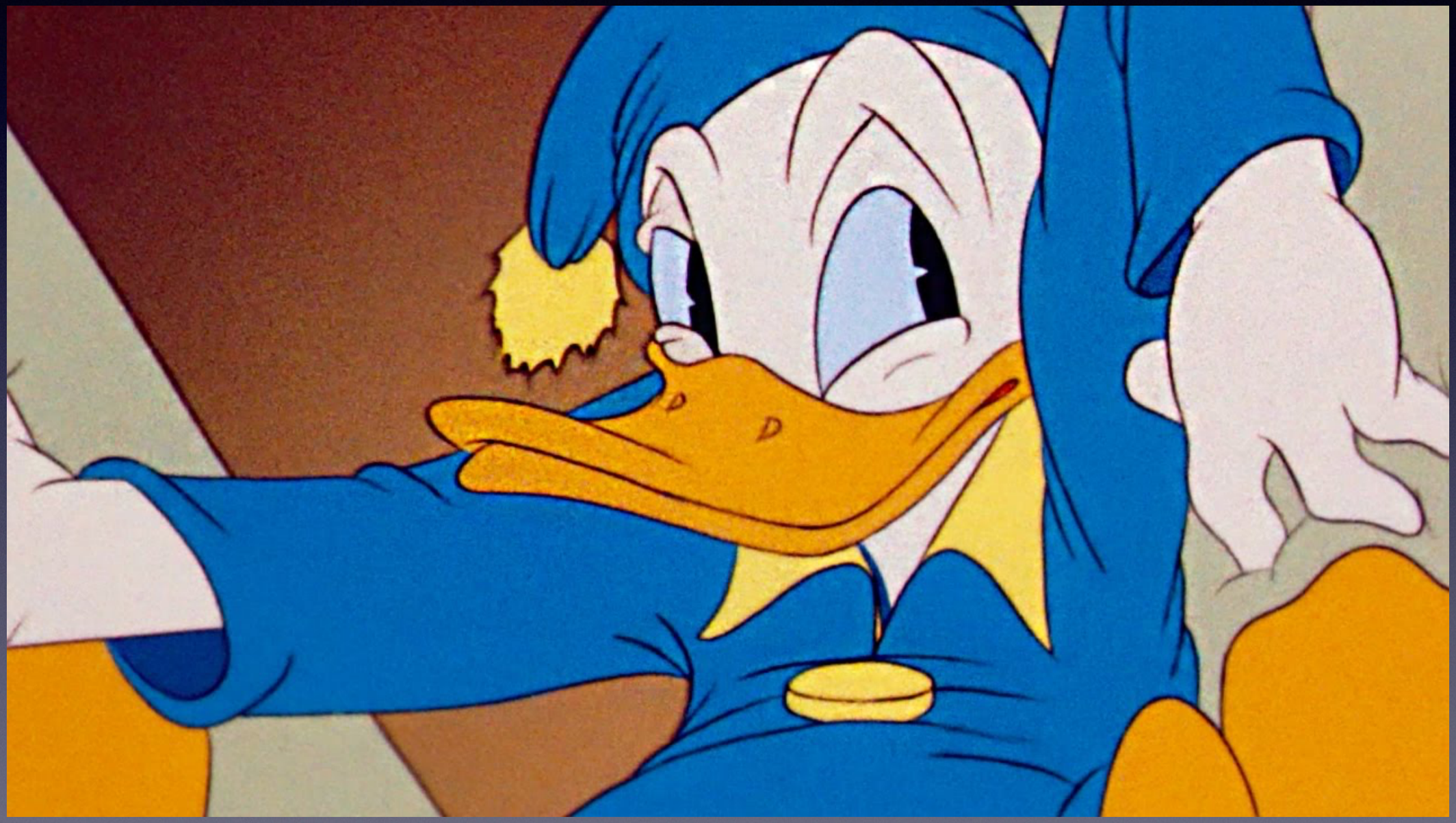
- Function: vars decl'd in function is not accessible outside after it terminates
- Loops: vars decl'd in a loop are only accessible within the loop scope

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

Liskov Substitution Principle



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>();

Iterators

- Interface
- Iterator <type> e = gettypeInstance().iterator();
- e.next();
- e.hasNext();
- e.remove();

Equality

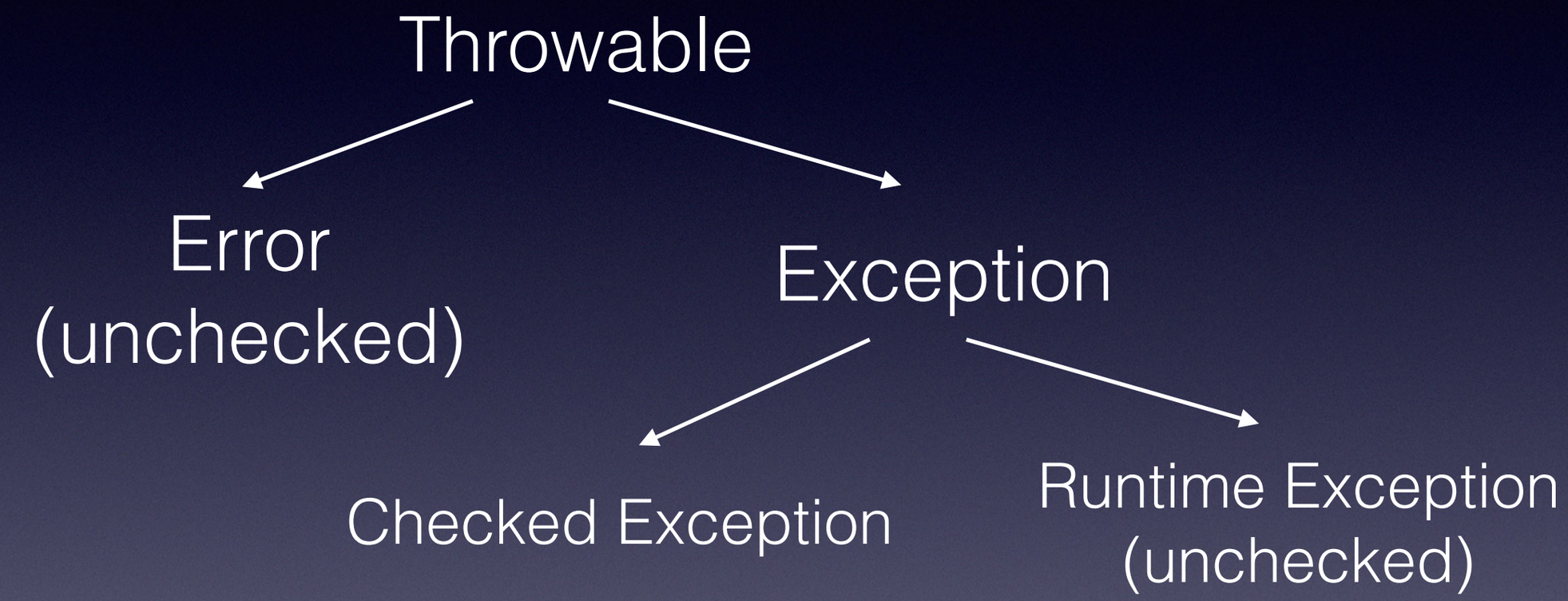
- Reflexive
- Symmetric
- Transitive
- Consistent

Junit test

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

- assertTrue
- assertEquals
  - may need to write your own assertEquals

Exception Inheritance



Exceptions

- Checked vs unchecked
- Inherit from Throwable
- throw new exceptionName(); // throwing an obj!

public class exception name extends Exception
{…}

…

throw new ExceptionName();

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

Dynamic vs Static binding

- Dynamic binding: Binding on runtime
  - uses object to resolve binding
- Static binding: Binding on compile time
  - uses Type (a class) to resolve binding
- Everything is dynamic binding except:
  - private
  - static
  - final
  - overloaded

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();
```

Refactoring

- To rework code, to make it better
- Code should still work after refactor
- Should be done when quality can be improved
- Do NOT refactor someone else's code unless they're no longer maintaining it (eg left the company)
- Steps:
  - Make small change
  - Run all test
  - If work -> next
  - else -> fix

Code Smells

- Inflexible design
- Duplicate code
- Long methods
- Big classes
- Big switch statements
- Long call chains
- Lots of null check
- un-encapsulated fields

Fixes

- Don’t use switch to differentiate diff type of obj
- Extra classes, interface, methods, etc
- Subclass, superclass
- Replace error codes with (custom) exceptions

Code Review

- Someone else reads your code and give you feedback

- dry code = no repeated code (good)

Things not covered but you should totally study

- UML & CRC Cards (!!important)
- Java Ant, Build file in XML
- JVM, Java compilation process
- Regular Expression in Java
- Agile vs waterfall
- Floating Point