Welcome

Anything that shows up in the powerpoint or was spoken about by Prof Banerjee may appear on the exam. My best bet is to memorize the powerpoints w o r d - f o r - w o r d.

By Fanng Dai

I stole notes from Banerjee's slide which gives Fodor credit so.. I guess I stole Banerjee's slide which were stolen from Fodor

Basics everyone should know

- Subclass
- Child class

- Parent class
- Base class
- Super class

- Extends
- Implements

- Abstract Classes
- Interface Classes
- Enum Classes
- Concrete Classes

- super()
- this() this.

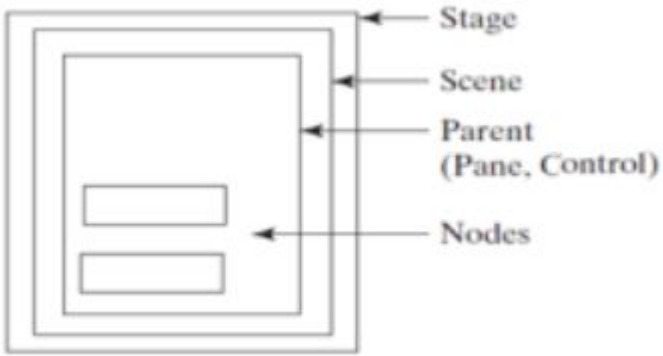
- Overriding
- Overloading

Good Quality Software

1. Correctness
2. Efficiency
3. Ease-of-use
4. Reliability
5. Maintainability
6. Modifiability and extensibility
7. Scalability

Structure of JavaFX

- Application: entry point for javaFX Application



Know what they do and the difference of btw them

- Event Handling
- Event Listeners
- Event Objects
- Event Targets
- Event Types

- Mouse Event
- KeyEvent

- Inner Class
- Lambda expressions

Anonymous Inner Class - I got these from some website

Inner class without a name is called anonymous inner class

1. Has no name
2. Can be instantiated only once
3. Is usually declared inside a method or a code block, a curly braces ending with semicolon
4. Is accessible only at the point where it is defined
5. No constructor, bc no name
6. Cannot be static

Conversions - Powerpoint type erasure

```
int a = 1;
Integer b = new Integer(1);      // boxing
Object c = b;                    // Widening Conversion
b = (Integer) c;                 // Type casting
int d = b.intValue();            // unboxing
Integer e = a;                   // autoboxing
int fanny = e;                   // auto-unboxing
```

Type-casting: narrows conversion

Child = new Parent(); // run-time error

Child = (Child) new Parent(); // compile-time error

So here, which ones throw run-time and which throws compile-time?

```
Person p = new Person();        p = (Person)new Student();
Student s = new Student();      p = (Student)new Student();
p = new Student();              s = (Person)new Person();
s = new Person();               s = (Student)new Person();
```

Generics Practice coding this

- Except for primitive types, any variable in Java is a reference (i.e., pointer) to an object
- Type Erasure: A mechanism that removes specific type info within the body of a generic class or method
- Operators >,<,>=,<= Use compareTo()
- Wildcards?
  - List<? extends Number> // To put into a list, descendant, subclass
  - List<? super Integer> // To add, ancestor, superclass

List & LinkedList of type object

List<Object> obj = (List<Object>) new ArrayLIst<String>(10);

Compile-time error bc list is an object

Object obj = new String(“S”); // This is valid why? Bc arrays are outdated

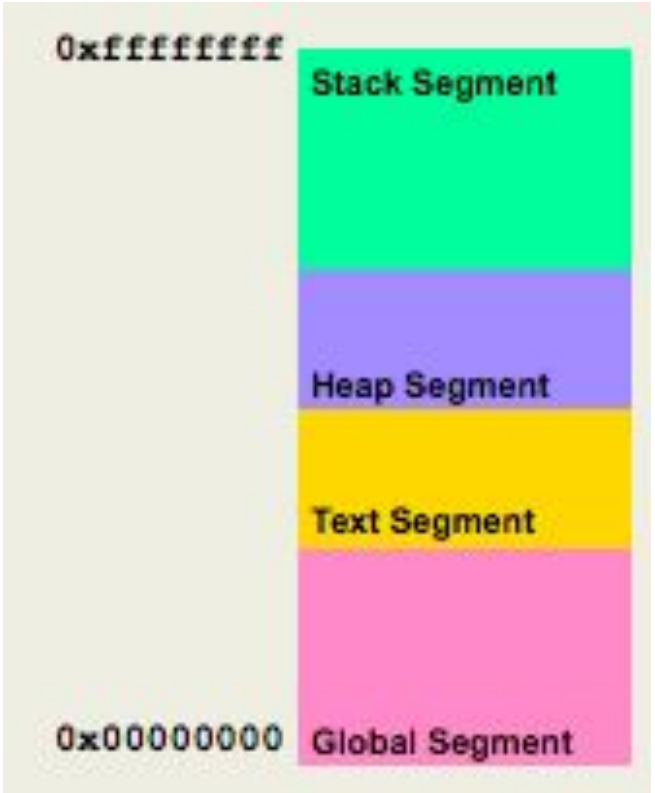
Multiple Bounds

Class/Interface A
Interface B
Interface C
Class D<T extends **A** & B & C> // This is ok

Class D<T extends **B** & A> // This is not ok. What error will it result in?

Stack - You should know this for CSE220 & 320 as well as this course

- Memory
- Text Segment
  - Program instructions
- Global Segment
  - static methods and variables
- Stack Segment
  - Temporary variables
- Heap Segment
  - New someObject() dynamic data



Comparable vs. Comparator

comparable : compares 2 given objects

comparator : compare a given object & this

Note: When implementing, must be the same!

public static<T extends Comparable<? Super T>> void sort(List<T> list)

public static<T> void sort(List<T> list, Comparator<? super T> c)

Apparent vs. Actual

Person p = new Student();

Person is the apparent type
Compiler cares about the apparent type
Determines what methods can be invoked on an object

Student is the actual type
Runtime cares about actual type
Determines which implementation of the method is called

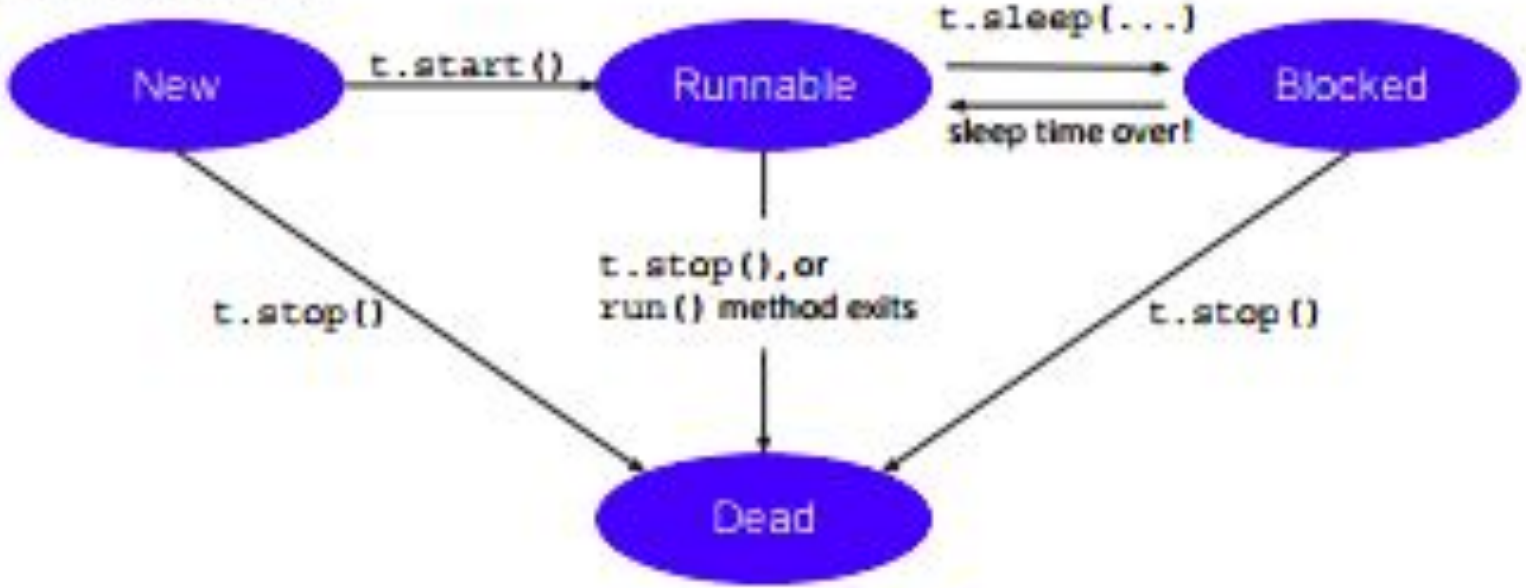
Multithreading

- Time Slicing
  - Able to stop & continue later
  - Begin task -> stop. Do something else -> continue task
- Atomic - synchronously
  - Cannot time-slice your program
  - Cannot be broken down into sub-transactions
  - Using the same clock
  - A reserved keyword used by a thread to acquire a lock
- Asynchronously
  - Not using the same clock
- Fixed-priority
  - Ranges from 1(MIN_PRIORITY) to 10(MAX_PRIORITY)
  - Default is 5(NORM_PRIORITY)
- Guard block: keeps checking for a condition to be true

Very important keywords for Multithreading when you almost typed multicycle lmao

- Liveness
  - Code is working. Everything is ok.
- Deadlock
  - Occurs when 2 threads are on lock & want to lock the other one
  - To solve, don't have a waiting thread lock other threads. Release lock before wait.
- Starvation
  - A situation where a thread is unable to gain regular access to shared resources & is unable to make progress
  - How I like to think of starvation. As an east asian, I can't eat with only one stick.
- Livelock
  - A thread acts accordingly to another thread & the relies on another
  - Too busy calling other threads that the thread cannot process itself.
- Race Condition

```
Thread t = new Thread();
```



Thread Methods

Note for using multithreading: Never trust it too much. It is not obedient

- **Thread.sleep** causes the current thread to suspend execution for a specific period (pause execution)
- **Thread.start** starts a new thread then calls the run() method
- **Thread.run** continues the thread form where it was left off
- **Thread.interrupt** indicates to a thread that it should stop its current task and do something else

Thread Methods (continued)

- **Thread.join** makes one thread wait for another to complete
- **Thread.stop** kills the thread & this thread will not be able to revive
- **Thread.notifyAll** wakes up all threads waiting on this objects intrinsic lock
- **Thread.notify** picks an arbitrary thread to wake (from this object)
- **Thread.wait** to suspend the current thread

Thread Methods (continued)

wait() must own your own intrinsic lock o.w. Throw IllegalMonitorStateException

aWait() to wait until a condition is ‘signaled’

signal() to wake up one waiting thread

SignalAll() to wake up all waiting threads

Unified Modeling Language (UML)

Class name -> variables -> construction & methods

Class A knows and uses Class B → Class B does not know

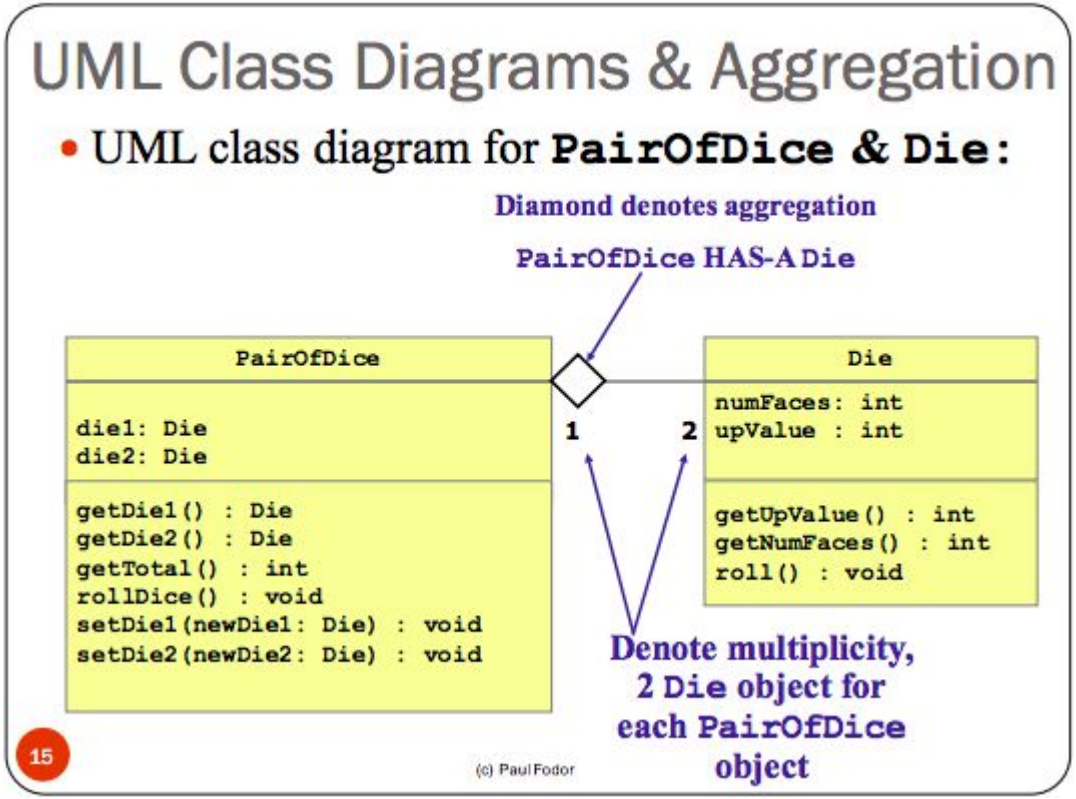
— The two classes have a relationship

+ public - private # protected

<<Interface>> or <<I>> {abstract} or *someClassNameThisInItalics*

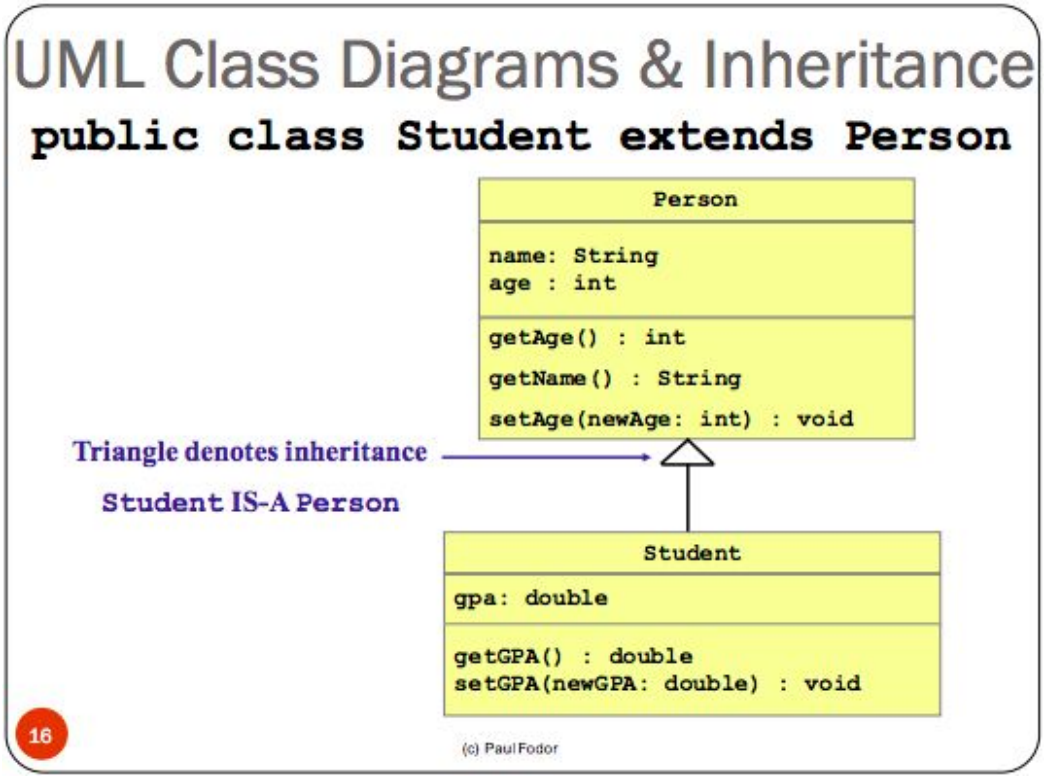
Aggregation

Cannot live without



Inheritance

extends



Creational Design Patterns

Design Pattern: A description of a problem and its solution that can apply to many similar programming situations

Telescoping Constructor

- Factory Pattern
- Singleton Pattern
- Builder Pattern
- Prototype

Behavioral Design Patterns

Identify common communication patterns between objects increase the flexibility in carrying out communication. Describes the object & classes interaction & divide responsibilities among themselves

- Chain of Responsibilities
- Strategy Pattern
- Template Pattern
- **Observer Pattern**
- Command Pattern
- **Iterator Pattern**
- Interpreter Pattern
- **Mediator Pattern**
- **Memento Pattern**
- State Pattern
- **Visitor Pattern**

Hmmm I wonder why some of them are in bold... hmmm

Basic Vocabulary

- Encapsulate
- Abstraction
- Polymorphism
- Primitive types
- GUI (Graphical User Interface)
  - AWT (Abstract Windows Toolkit)
  - Swing replaces awt more flexible

Some notes - facts

- Except for object, every class in java has only one superclass
- Java is strictly PASS-BY-VALUE
  - An object itself is never passed by value
  - References to objects are passed by value
  - Primitive are passed by value
- An interface is a reference data type
- Every object has an intrinsic lock associated with it

Some Question

Many questions are random. In this case, you would have to use process of elimination and guess your way.

1. What is not inherited from a subclass?
2. When overriding, what does the new method do to the superclass's method?
3. The state of Java Object is stored in its _____.
4. java.lang.Object is a what class? (Some questions are random like this one...)
5. What is binding properties? Give me a simple definition.

Some More Question

1. What happens when a class is declared final?
2. What happens when a variable is declared final?
3. What happens when a variable is declared static?
4. What do we call the tree structure that manages and stores all the content in a web page?

These are some bookmarked websites from when I took this course. Ofc, some of them may be final material... idk
Good luck with your exam :)

http://www.javamadesoeasy.com/2015/10/threadmulti-threading-quiz-in-java-mcq.html#top

https://www.indiabix.com/java-programming/questions-and-answers/

https://sourcemaking.com/design_patterns

https://docs.oracle.com/javase/tutorial/reflect/

Welcome

final

By Fanng Dai

Structural Design Pattern

Describe the structure of a software

- Decorator Pattern
- Adapter Pattern
- Facade Pattern
- Flyweight Pattern
- Bridge Pattern

Test-driven Development

- Top-down development > Bottom-up development

- Unit testing
- Integration testing
- Regression testing
- System testing
- Black-box testing
- Glass-box testing
- Boundary-value Analysis

IO Testing

IO components are usually either read from an inputstream or written to an outputStream

Ex: read from file

Using annotation

@Test(expected = IllegalArgumentException.class)

// test fails if ^ is not thrown)

Designing with Exceptions

- Brute force (System.out.print)
- Robust

Annotations

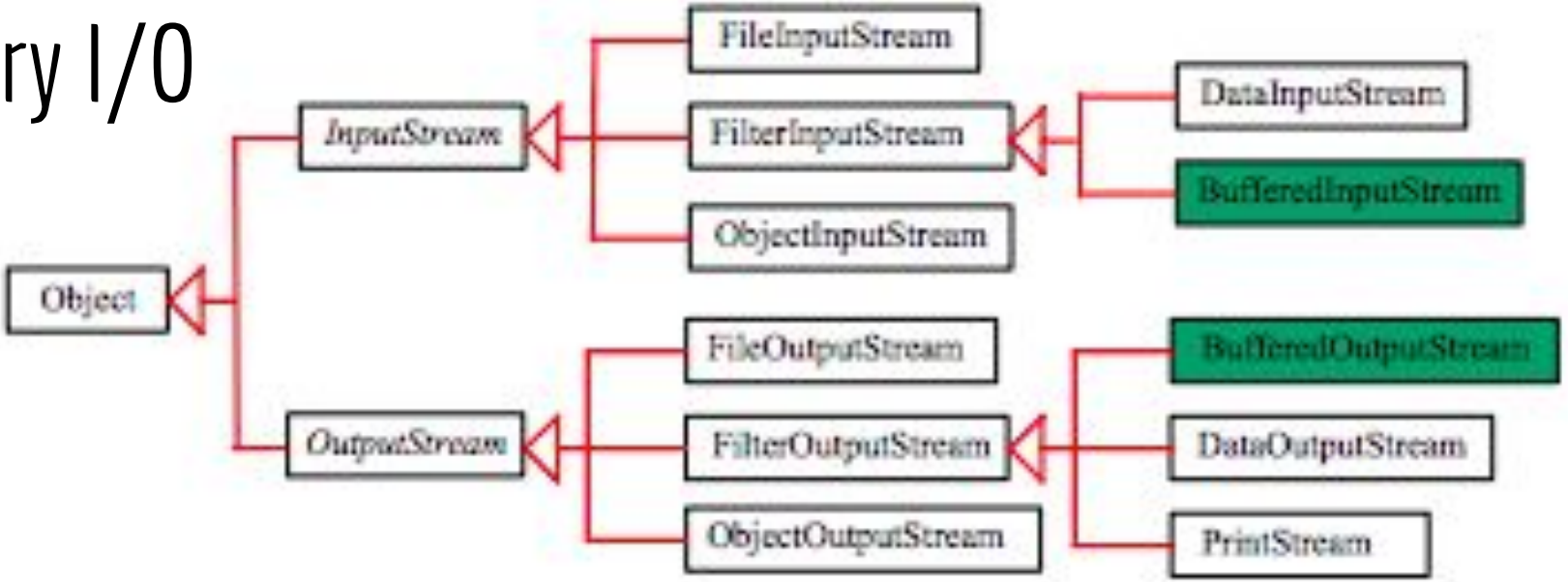
@Deprecated

@Override

@SuppressWarning

@Document**ed**

Binary I/O


Object
InputStream
FileInputStream
FilterInputStream
ObjectInputStream
DataInputStream
BufferedInputStream
OutputStream
FileOutputStream
FilterOutputStream
ObjectOutputStream
BufferedOutputStream
DataOutputStream
PrintStream

Reflection

Type introspection: ability of a program to examine the type & properties of an object at runtime ex: instanceof

Boolean b;

Class c = b.getClass(); // compile time error

Class c = Class.forName(“full.path.MyObject”);

Class c = Double.TYPE; // Primitive wrappers

Unchecked Operations

getMethod() causes a typical unchecked conversion to solve

1. Modify the declaration to include generic type

   Class <?> c = warn.getClass();

2. Warning @SuppressWarnings(“unchecked”)

   This is bad bc it hides bugs

A function is used as a Function object by denoting it by the :: operator
If C is a class & m is a function in that class implemented as a static method, you would refer to its corresponding function object as c::m

Interface java.util.function.Function<T,R>
T - input argument
R - return type

Lambda expression
BiFunction<Integer, Integer, Integer> add = (a,b) -> a + b;
First 2 are arguments last integer is return type

Software Engineering Terms

- Scalability
  - I.e., the number of users or the amount of data processed
- Robustness
- Extensibility

The End

Good luck on your final(s)~~~