Ch. 8 Objects & Classes: Deeper Look 10/23/17

* public String to String *

State - values of an instance variable @ some time

- int values get values 0.

- any java file contains only 1 public class

- Access Rules

	World	Package	class
public	✓	✓	✓
package private		✓	—
private			✓

overloaded constructer

- build an object, there's a reference (this)

- this. variable → instance variable

- only 1 public class / .java file

- overload constructors — call another constructor

- no- arguement constructor

- every class needs to have 1 constructor.

- java provides defaults

- mutator methods (set methods) - change state
- accessor methods (Get ") - (query Methods
- validity checking
- predicate methods - test condition true or false

- Composition - class can have references to object to other classes of members
(AKA - has-a relationship)

- Reference, like a memory address

to String
↳ captures entire state of object into String
↳ values of all instance variables
@ a particular moment.

enums declaration
↳ countable

- final - can't change
static - not property of object, but property of class
- values - returns arrays of values.

- Garbage Collection - program open for long period of time, free storage.
↳ looks for things currently not being used

10/27/17

- static field - called class variable - is used in such cases
- static var. have class scope
- static method cannot access a class's instance variable / method
- if static var. not initialized, compiler assigns default value. 0 for int.
- String Objects in Java are Immutable - state can't be changed.
- eligible objects - don't have a valid reference.
- principle of least priviledge: granted privi edge code needs to accomplish its task, but no more.
- list final, for least priviledge.
- package access - no access modefier is specified for a method or variable.

example:

```
public class Pokemon
{   private string type;
    "      int health,
    "      "   power;


    public Pokemon (string type, int health, int power)
    {   this.type = type;
        ".health = health,
        ".power = power


    }
    pubic String getType()
    { return this.type; }
    public int getHealth()
    { return this.Health(); }
    pubic int getPower
    { return this.power, }

    pubic String toString
    { string val= "",

      val +=
```

```
public void attack (Pokemon opponent)
{ opponent.getHit(this.power)

public void getHit (amtOfHit)
    health -= amtOfHit,
    if (health >= amtofHit)
```

```
pub class PokemonRunner
{
    main Method {

Pokemon pikatu = new Pokemon("Pikatu", 35, 5);
   "    squirtle = "  Pokemon("squirtle", 45, 6);
   "    charman = "  Poken("charman", 4, 1);

System.out.println("first: " + pikatu);
   "     "     "     ("2nd " + squirtle);
   "     "     "     ("3rd " + charman);


charman.attack(pikatu);
pikatu.attack(charman);
```

Chapter 9: OOP - Inheritance

10/30/17

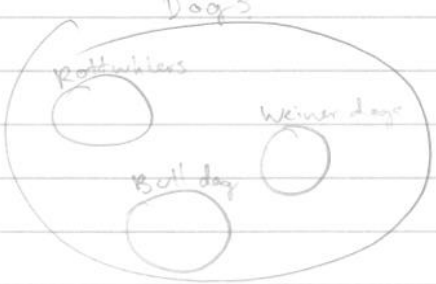
- new class is created aquiring an existing class's members.
- existing class → superclass
- New class → subclass (concrete)
- inheritance referred to as specialization

ex: Animal → mammal → human → student → CSCI student

→ concrete

← abstract

- subclass can have methods
- direct superclass - superclass from which subclass explicitly inherits.
- indirect superclass - any class above the direct superclass in class hierarchy.
- every class in Java extends (or "inherits from") objects.
- Java supports only single inheritance.
- is-a → extends
- " " has a relationship



- is-a relationship → inheritance
- Has-a relationship → Composition
- the subclass inherits everything, Except constructor

ex: Public class extends Dog {

- in a subtype constructor, the VERY First thing that happens is a call to the supertype's constructor

- use super to call superclass

- subclass can override
  ↳ signatures must match

11/1/17

- subclass can override superclass method w/appropriate implimentation.

- protected is an intermediate access between public & private

  - gives access to any subclass method
  - " " " every package

if superclass is private, members are hidden from subclass

ex: super.bark();
↳ invoks dog's bark method.

- to string - packages up current string & sends it to whoever asks.

- @Override annotation to indicate that following method declaration should override existing superclass

- more subclass has access, it gets fragile & brittle.

Ch. 10

11/3/17

- polymorphism - enables you to "program in the general" rather than "program the specific."

ex: `Dog leo = new Poodle ("Leo", 14.0, 33.0);`
all poodles are dogs

static type of leo is dog

static type	dynamic type
var. declaration	depends on type of object assigned
fixed	Not fixed
set @ compile time	set @ run time
supertype	supertype or subtype

* instance of *
  ↳ boolean expression

upcasting
  ↳ always safe, goes to more general type

downcasting
  ↳ checking, using instance of

interface - defines a new type, doesn't contain data. Does NOT have method implementations.

→ defines method signatures

interface not a class, but defines a type.
ex: public interface DogInterface
" class Dog implements DogInterface

- main thing interface has is abstract methods

→ public abstract void bark();

→ label class abstract