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

P C Room {

```
private String description;
"       Room roomToNorth;
"       "    "  "S;
"       "    "  "E;
"       "    "  "W;

public Room(String description) {
  this.description = description;
  "    .RomtoNorth= null;
  "       S
  "       E
  "       W
}

P       String getDescription(){
    return description;
}
P   Room getRoomN(){
    return this.RoomN(S)(E)(W);
{

P   void setroomtoNorth(Room room){ // each direction
    this.roomToNorth = room;

@Override
public String toString(){
  String val = "youre currently in " + getDescription();
       " += "\n Exits are: ";
  if (getRoomToNorth() != null);          // same for S, E, W
    val += "N";
  return val;
```

P C RoomTester

```
public static void main (String[] args)

Room kitchen = new Room (" ")
 "   bed
 "   bath

bedroom.setRoomNorth(bathroom)
 bath   . " "    South(bed)
 bed    . " "    E    (kitchen)
 kitchen. " "    W   (bed)

  Room current = kitchen
// print (" current room: " + currentRoom.getDescription())
// "  (current room);
// going north
  currentRoom = currentRoom.getRoomNorth
  if (currentRoom.getRoomNorth() != null)
   else
     // print ("Can't go that way!")
```

Packman Game

11/8/17

Contains Main
Build GameBoard
Start Game "running"

Class GameBoard

has-a packeman has-a fruit
has 4 ghost has walls
has many dots
has 4-powerDots

Class Packman

has-a direction, has-a speed, has-a position

Class Ghost

has-a direction, has-a speed, has-a position

Class Dot

has-a position

→ Class PowerDots
has-a position

Class Fruit

has-a position

Class Wall

Class Point2D

```
class Point2D{
    private int[] coords;

public Point2D() {
call constructor → this(0,0);   }

public Point2D(int x, int y) {
    coords = new int[2];
    "  [0] = x;
    "  [1] = y;

public int getX {
  return coords[0];
```

same w/ y

```
Public void setX(int newX){
  coords[0] = newX;
```

same w/ y

② Override

```
Public string toString{
  return "x: " + coords[0] + "y: " + coords[1];
```

Review

11/10/17

Arrays:

array - collect to store data of same type

↳ int, string, ...

↳ reference types

```
int [] myVals;      ⇐ array not built.

int [] myVals = new int [100];
```

access it by name of array.

int [][] -> 2 dim. Array

enhanced for-loop

ex: String studentNames = new String [10];

```
for ( String Name : StudentNames)
    System.out.println(name);
```

array initializer expression

```
int [] arr = {1, 2, 3, 4};
```

Objecti

↳ instance of an new data type that you define

class - blueprint for objects

act of building an object is instantiation

State - all values of instance variables @ any moment.

- label instance variables private b/c of encapsulation

- encapsulation
  - ↳ principle of least privelege

public method
- ↳ called outside class

Objects state change are mutable
" " don't change are immutable

final - value cannot be change
don't provide setter methods

2 methods of same name = overloaded method

this. is "me"

```
Dog fifi = new Dog("fifi");
"  butch = "  " ("butch");
"  spike = fifi;
```

	T	F
fifi == butch		✓
butch == spike		✓
spike == fifi	✓	

→ Inheritence:

class is composed of other classes

composition → has-a relationship
is-a relationship

Scottish Terrier extends terrier

11/6/17

ex: private in Dog

callin terrier → table it protected

class can extend only one class

↳ single inheritance

Dog Jack = x;

↑ you can put subtypes of dog

↑ upcasting is safe
↓ downcasting " dangerous

label method so it can't be overridden

" class " " " " extended
↳ can't build subtypes of it