Array Activities:

* Make an array with String elements that holds the names of the Seven Dwarves

String[] dwarves = {“Sleepy”, “Bashful”, “Doc”, “Happy”, “Grumpy”, “Sneezy”, “Dopey”};

* Make an array of length 5 where you don’t know the values of the elements

int[] myArray = new int[5];

* + Now fill the array with 5 random values:

myArray[0] = 3;

myArray[1] = 4;

myArray[2] = 56;

myArray[3] = 23;

myArray[4] = 605;

* Make an array of length 100 where each element has the same value

int[] someArray = new int[100];

for (index = 0; index < someArray.length; index++)

someArray[index] = 1;

* Print the elements in an array to the screen, on the same line, separated by tabs

for (i = 0; i < someArray.length; i++)

System.out.print(someArray[i] + “\t”);

* Print the length of an array to the screen

System.out.println(myArray.length);

* Make a 3x? ragged array of random lengths

double[][] raggedArray = new double[3][];

raggedArray[0] = new double[2];

raggedArray[1] = new double[8];

raggedArray[2] = new double[5];

* Make a 10x10 multidimensional array and fill every element with the same value
* Make a ragged array that steps from 1 element per row, to 2 elements per row, to 3 elements per row and so on and fill every element with the same value

Inheritance Activities:

* Get everyone to list their favourite PC, Console or Tabletop game.
* Ask student to draw inheritance diagram.



* Discuss difference between ‘Has-A’ and ‘Is-A’. A Playstation Game is a Console Game and is a Game. A Card Game is a Tabletop Game and is a Game. A Tabletop Game has a Table.
* Ask for class participation in outlining ‘Game’ superclass.

public class Game{

private String rule;

private int numPlayer;

public Game(){

rule = “No Rules”;

numPlayer = 0;

}

public Game(String aRule, int numPlayer){

rule = aRule;

this.numPlayer = numPlayer;

}

public Game(Game originalGame){

rule = originalGame.rule;

numPlayer = originalGame.numPlayer;

}

public String getRule(){

return rule;

}

public int getNumPlayer(){

return numPlayer;

}

public setRule(String rule){

this.rule = rule;

}

public setNumPlayer(int num){

numPlayer = num;

}

public String toString(){

return (“Rules: “ + rule + “ Number of players: “ + numPlayer);

}

public boolean equals(Object other){

if(other == null)

return false;

else if (getClass() != other.getClass())

return false;

else{

Game otherGame = (Game)other;

if((numPlayer == otherGame.numPlayer) && (rule.equals(otherGame.rule)

return true;

else

return false;

}

}

}

* Split up into groups and create classes for subclasses

public class Tabletop extends Game{

private Table tableUsed;

public Tabletop(){

super();

tableUsed = new Table(0);

}

public Tabletop(String rules, int players, Table theTable){

super(rules, players);

tableUsed = new Table(theTable);

}

public Tabletop(Tabletop originalTabletop){

super(originalTabletop);

tabledUsed = new Table(originalTabletop.tableUsed);

}

public getTable(){

return new Table(tableUsed);

}

public setTable(Table newTable){

tableUsed = new Table(newTable);

}

public String toString(){

return(super.toString() + “ Table: “ + tableUsed.toString());

}

public boolean equals(Object other){

if(other == null)

return false;

else if (getClass() != other.getClass())

return false;

else{

Tabletop otherTabletop = (Tabletop)other;

if((getNumPlayer() == otherTabletop.getNumPlayer()) && (getRule().equals(otherTabletop.getRule()) && (tableUsed.equals(otherTabletop.tableUsed)))

return true;

else

return false;

}

}

}

Polymorphism Activities:

* Using the inheritance diagram from above, discuss whether the following declarations are legal:
  + PC a = new PC(); (Valid)
  + Game b = new PC(); (Valid)
  + Card c = new Tabletop(); (Invalid)
  + Object d = new Card(); (Valid)
  + Card e = d; (Requires downcasting (Card e = (Card) d;) which is not always safe and throws a runtime exception if the instance to be downcasted does not belong to the correct subclass)
* Using the inheritance diagram from above and the declarations provided, discuss whether the following statements will print out true or false:
  + Game g = new Game();
  + Console c = new Console();
  + Game z = new Console();
  + System.out.println(g instanceof Game); (true)
  + System.out.println(g instanceof Console); (false)
  + System.out.println(c instanceof Console); (true)
  + System.out.println(c instanceof Game); (true)
  + System.out.println(z instanceof Game); (true)
  + System.out.println(z instance of Console); (true)
* If the method ‘greeting()’ is defined for both the ‘Birthday’ and ‘AdultBirthday’ classes, which method will run for the given code:
  + Birthday happy;

happy = new AdultBirthday(“Joe”, 39);

happy.greeting();

* + The ‘greeting()’ method defined in the ‘AdultBirthday’ class because that is the type of the object referred to by ‘happy’.
* The class ‘Rodent’ has a child class ‘Rat’ and another child class ‘Mouse’ and the class ‘Mouse’ has a child class ‘PocketMouse’. Given the declarations below, which of the following will cause a compiler error:
  + Rodent rod;

Rat rat = new Rat();

Mouse mos = new Mouse();

PocketMouse pkt = new PocketMouse();

* + rod = rat; (valid)
  + rod = mos; (valid)
  + pkt = null (valid)
  + pkt = rat (invalid)
* Using the same situation as above, which of the following array declarations is correct for an array that is expected to hold up to 10 objects of types ‘Rat’, ‘Mouse’, and ‘PocketMouse’
  + Rat[] array = new Rat[10]; (incorrect)
  + Rodent[] array = new Rat[10]; (incorrect)
  + Rodent[] array = new Rodent[10]; (correct)
  + Rodent[10] array; (incorrect)
* What is the output of the following code:
  + class Hotel{

public int bookings = 2;

public void book(){

bookings++;

}

}

public class SuperHotel extends Hotel{

public void book(){

bookings--;

}

public void book(int size){

book();

super.book();

bookings += size;

}

public static void main(String args[]){

SuperHotel s = new SuperHotel();

s.book(2);

System.out.print(s.bookings);

}

}

* + Output = 4