Interface List<E>

add(E e)	appends the specified element to the end of this list
add(int index, E element)	inserts element at specified position
addAll(Collection E)	adds all elements to end of list
clear()	removes all elements from the list
contains(Object o)	returns true if it contains object
containsAll(Collection c)	returns true if all elements of collection are contained
equals(Object o)	compares specified object within list for equality
get(int indedx)	returns element at specified position
hashCode()	returns hashCode value
indexOf(Object o)	returns first index of specified object
remove(int index)	removes element at specified position
set(int index, E element)	replaces element at specified index
toArray()	returns an array containing all the elements

-ordered collection
-user has control over where in the list each element is inserted
-ALLOW DUPLICATE ELEMENTS

Definitions

hashCode()

> if two objects are equal according to .equals() must have same hashcode

Interface Set<E>

add(E e)	Adds the specified element if not already present
addAll(Collection c)	Adds all the elements if not already present
clear()	Removes the elements from this set
contains(Object O)	Returns true if specified element is contained
equals(Object O)	Compares specified Object
hashCode()	returns the hash code value for the set
remove(Object O)	removes the specified object if present
size()	returns number of elements
toArray()	Returns an array containing the elements of the set

E - the type of elements maintained by the set
-contains no duplicate elements

Class HashSet<E>

add(E e)	adds the element if not already present
clone()	shallow copy, elements themselves not cloned
remove(Object o)	removes object if present
size()	returns number of elements

-no duplicates
-no guarantee as to the iteration order of the set

Class String

charAt(int index)	returns the character at index
compareTo(String anotherString)	compares two strings alphabetically (caps matter)
contains(Char c)	returns true only if the character is contained
equals(Object anotherObject)	compares to another object
hashCode()	returns hashCode for the string
length()	returns the length of the string - counts each character including spaces
split(String regex)	splits the string at all point of the regex
toCharArray()	converts to a character array
toLowerCase()/toUpperCase()	converts all characters to lower/upper case

-basically a character array. Below are equivalent
String str = "abc";
char data[] = {'a', 'b', 'c'};

Class Array

asList(...)	returns a fixed size list
equals()	returns true if the objects compared are equal to one another
hashCode()	returns a hashcode based on the elements of the array
toString()	returns a sting representation of the array

Class ArrayList<E> and Class LinkedList<E>

ArrayList
-resizeable array implementation of the List interface
-has just about all the methods defined in the List interface

LinkedList
-implements nearly all List methods
-traverse list from beginning or end

Circles Country

```
public class CirclesCountry {
/**
* Returns true if a point is inside a circle and
* returns false otherwise.
* @param x is x-coordinate of point
* @param y is y-coordinate of point
* @param cx is center of circle x-coordinate
* @param cy is center of circle y-coordinate
* @param r is radius of circle
* @return true if (x,y) is inside circle, returns
* false if (x,y) is on or outside circle
*/
public boolean isInside(int px, int py, int cx, int cy,
int r){
double dist; //distance from the center of the circle
dist = Math.pow((px - cx), 2) + Math.pow((py - cy),
2);
return dist < r*r;
}

public int leastBorders(int[] x, int[] y, int[] r,
int x1, int y1, int x2, int y2) {
int crosses = 0;
for(int k = 0; k < x.length; k += 1){
if (isInside(x[k], y[k], x1, y1, r[k]) && !
isInside(x[k],y[k], x2, y2, r[k])) {
crosses += 1;
}
if (isInside(x[k], y[k], x2, y2, r[k]) && !
isInside(x[k],y[k], x1, y1, r[k])) {
crosses += 1;
```

Circles Country (cont)

```
}
}
return crosses;
}
}
```

Body

```
public class Body {
*/declares all the instance variables as private
doubles and String
public Body(double xp, double yp, double xv, double
yv, double mass, String filename) {
myXPos = xp;
myYPos = yp;
myXVel = xv;
myYVel = yv;
myMass = mass;
myFileName = filename;
}
/**
* Creates a body by copying parameters from another
body
* @param b
*/
public Body(Body b) { //FIXME
myXPos = b.getX();
declares the rest of the instance variables with .get
methods
}
/**
          all the get methods just return the
corresponding instance variable
        **/
public double calcDistance(Body b) {
return Math.sqrt((Math.pow(myXPos - b.getX(), 2.0)+
Math.pow(myYPos - b.getY(), 2.0)));
}
Calculates the force exerted on this body by the body
specified in the parameter
```

Body (cont)

```
public double calcForceExertedBy(Body p) {
double force;
double G = 6.67e-11;
force = (GmyMass p.getMass()) /
Math.pow(p.calcDistance(this), 2.0);
return force;
}
/**
* Calculates the X component of the force
*/
public double calcForceExertedByX(Body p) {
double dx;
double Fx;
dx = myXPos - p.getX();
Fx = p.calcForceExertedBy(this) *
(dx/(p.calcDistance(this)));
return -Fx;
}
* calculates the Y component of the net force
public double calcForceExertedByY(Body p) {
/* same as the X just different variables
}
/**
* calculates the net force in the x direction exerted
on a body
*/
public double calcNetForceExertedByX(Body [] bodies) {
double sum = 0;
for (Body b : bodies) {
if (!b.equals(this)) {
sum += b.calcForceExertedByX(this);}}
return -sum;}
/**
* calculates the net force in the y direction exerted
on a body
```

Body (cont)

```
*/
public double calcNetForceExertedByY(Body [] bodies) {
double sum = 0;
for(Body b : bodies) {
if (!b.equals(this)) {
sum += b.calcForceExertedByY(this);
}
}
return -sum;}
/**
* updates the velocity, position, and time variables
*/
public void update(double deltaT, double xforce,
double yforce) {
double ax;
double ay;
ax = xforce / this.getMass();
ay = yforce / this.getMass();
myXVel = myXVel + deltaT*ax;
myYVel = myYVel + deltaT*ay;
myXPos = myXPos + deltaT*myXVel;
myYPos = myYPos + deltaT*myYVel;
}
/**
* shows a non-moving image for each body in the
simulation
*/
public void draw() {
StdDraw.picture(myXPos, myYPos,
"images/"+myFileName);
}
}
```

Anonymous

```
public class Anonymous {
public int howMany(String[] headlines, String[]
messages) {

// counts[ch] is # occurrences of ch in headlines
        int counts[] = new int[256];
        for(String s : headlines) {
         for(char ch : s.toLowerCase().toCharArray())
{
         if (ch == ' ') continue;
         counts[ch] = counts[ch] + 1;
         }
        }

        int total = 0;
        for(String s : messages) {
         int[] mess = oneCount(s);
         if (enoughLetters(mess,counts)) {
         total += 1;
         }
        }
        return total;
  }
private boolean enoughLetters(int[] mess, int[]
allLetters) {
boolean canCreate = true;
     for(char ch='a'; ch <= 'z'; ch += 1) {
     // do we have enough letters to create message?
     if (mess[ch] > allLetters[ch]) {
     return false;
     }
     }
     return true;
}
```

NBody

```
public class NBody {

/**
* Read the specified file and return the radius
* @param fname is name of file that can be open
* @return the radius stored in the file
* @throws FileNotFoundException if fname cannot be
open
*/
public static double readRadius(String fname) throws
FileNotFoundException {
Scanner s = new Scanner(new File(fname));
double radius;
radius = s.nextInt();
radius = s.nextDouble();

s.close();
return radius;
}

/**
* Read all data in file, return array of Celestial
Bodies
* read by creating an array of Body objects from data
read.
* @param fname is name of file that can be open
* @return array of Body objects read
* @throws FileNotFoundException if fname cannot be
open
*/
public static Body[] readBodies(String fname) throws
FileNotFoundException {

Scanner s = new Scanner(new File(fname));

// TODO: read # bodies, create array, ignore radius
int nb = 0; // # bodies to be read
nb = s.nextInt();
s.nextDouble();
Body [] bodiesArray = new Body [nb];

for(int k=0; k < nb; k++) {
double xPos;
```

NBody (cont)

```
double yPos;
double xVel;
double yVel;
double mass;
String fileName;

xPos = s.nextDouble();
yPos = s.nextDouble();
xVel = s.nextDouble();
yVel = s.nextDouble();
mass = s.nextDouble();
fileName = s.next();

bodiesArray[k] = new Body(xPos, yPos, xVel, yVel,
mass, fileName);
}

s.close();
return bodiesArray;
}

public static void main(String[] args) throws
FileNotFoundException{
double totalTime = 157788000.0;
double dt = 25000.0;

String fname= "./data/planets.txt";
if (args.length > 2) {
totalTime = Double.parseDouble(args[0]);
dt = Double.parseDouble(args[1]);
fname = args[2];
}

Body[] bodies = readBodies(fname);
double radius = readRadius(fname);

StdDraw.setScale(-radius, radius);
StdDraw.picture(0,0,"images/starfield.jpg");

for(double t = 0.0; t < totalTime; t += dt) {

// TODO: create double arrays xforces and yforces
```

NBody (cont)

```
// to hold forces on each body
double xForces[] = new double [bodies.length];
double yForces[] = new double [bodies.length];

// TODO: loop over all bodies, calculate
// net forces and store in xforces and yforces
for (int k = 0; k < bodies.length; k++) {
xForces[k] =
bodies[k].calcNetForceExertedByX(bodies);
yForces[k] =
bodies[k].calcNetForceExertedByY(bodies);

}

// TODO: loop over all bodies and call update
// with dt and corresponding xforces, yforces values
for(int k = 0; k < bodies.length; k++) {
bodies[k].update(dt, xForces[k], yForces[k]);
}

StdDraw.picture(0,0,"images/starfield.jpg");

// TODO: loop over all bodies and call draw on each
one
for (int k = 0; k < bodies.length; k++) {
bodies[k].draw();
}

StdDraw.show(10);
}

// prints final values after simulation

System.out.printf("%d\n", bodies.length);
System.out.printf("%.2e\n", radius);
for (int i = 0; i < bodies.length; i++) {
System.out.printf("%11.4e %11.4e %11.4e %11.4e %11.4e
%12s\n",
bodies[i].getX(), bodies[i].getY(),
bodies[i].getXVel(), bodies[i].getYVel(),
bodies[i].getMass(), bodies[i].getName());
}
}
```

NBody (cont)

```
}
```

MemberCheck

```
public class MemberCheck {
public String[] whosDishonest(String[] club1,
String[] club2,
String[] club3) {
Set<String> result = new HashSet<>(); //making these a
set ensures that there's no duplicates
Set<String> s1 = new HashSet<>(Arrays.asList(club1));
Set<String> s2 = new HashSet<>(Arrays.asList(club2));
Set<String> s3 = new HashSet<>(Arrays.asList(club3));
result.addAll(crossover(s1, s2));
result.addAll(crossover(s1, s3));
result.addAll(crossover(s2, s3));
String[] finalList = result.toArray(new String[0]);
Arrays.sort(finalList);
return finalList;
}
private Set crossover(Set club1, Set club2) {
Set<String> cloned = new HashSet<>();
cloned.addAll(club1);
cloned.retainAll(club2);
return cloned;
}
}
```

Queue

add(Element e)	inserts the specified element into the queue
remove()	calls and removes the head of the queue

PriorityQueue

Note*:	Head of the queue is the least element
add(Element e)	inserts specified element into priority queue
contains(Element e)	returns true if PriorityQueue contains specified element
remove(Object o)	removes single instance of the object
size()	returns number of elements in the PriorityQueue
peek()	retrieves but doesn't remove the head
poll()	retrieves and removes the head

Collections

Collections.sort(ArrayList)	sorts lowest to highest
Collections.reverse(ArrayList)	reverses the order

Arrays

Arrays.sort(int[] OR String[])	sorts it from smallest to largest or alphabetically first to last

Random Commands

Integer.parseInt(String s)	converts string to int
.split(" ")	
.substring(start index, end index)	Includes start index, not end index

Scanner

close()	closes the scanner
hasNext()	returns true if has another token
hasNextDouble()	if it has another double, there are other - same for float, int, line, long
next()	returns the next token

Scanner (cont)

nextDouble()	returns the next double token - works for float, int, line, long
reset()	resets the scanner