**Composition**

Author test result:

Tan Ah Teck (m) at teck@nowhere.com

Tan Ah Teck (m) at teck@somewhere.com

name is: Tan Ah Teck

gender is: m

email is: [teck@somewhere.com](mailto:teck@somewhere.com)

\*To String method prints some content when the system ask to print itself

Book test result:

Tan Ah Teck (m) at ahTeck@somewhere.com

'Java for dummies' by Tan Ah Teck (m) at ahTeck@somewhere.com

'Java for dummies' by Tan Ah Teck (m) at ahTeck@somewhere.com

name is: Java for dummies

price is: 8.88

qty is: 88

author is: Tan Ah Teck (m) at ahTeck@somewhere.com

author's name is: Tan Ah Teck

author's email is: ahTeck@somewhere.com

author's gender is: m

'Java for more dummies' by Peter Lee (m) at peter@nowhere.com

\*Here if we want the author’s name we have to go into two layers both

by entering method in author’s class: dummyBook.getAuthor().getName()

Point Class:

import java.lang.Math;

public class Point {

// Private variables

private int x; // x co-ordinate

private int y; // y co-ordinate

// Constructor

public Point (int x, int y) {

this.x = x;

this.y = y;

}

public Point(){}

// Public methods

public String toString() {

return "Point: (" + x + "," + y + ")";

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public void setX(int x) {

this.x = x;

}

public void setY(int y) {

this.y = y;

}

public void setXY(int x, int y) {

this.x = x;

this.y = y;

}

public int[] getXY(){

int[] xy= {x, y};

return xy;

}

public String ToString(){

return "(" + x + "," + y +")";

}

public double distance(Point end) {

return Math.sqrt((x-end.x)^2+(y-end.y));

}

}

Line test result:

Circle[center=Point: (0,0),radius=1.0]

Circle[center=Point: (1,2),radius=3.3]

Circle[center=Point: (4,5),radius=6.6]

Circle[center=Point: (11,12),radius=13.3]

center is: Point: (11,12)

radius is: 13.3

Circle[center=Point: (21,22),radius=13.3]

center's x is: 21

center's y is: 22

Circle[center=Point: (31,32),radius=13.3]

center's x is: 31

center's y is: 32

area is: 555.72

circumference is: 83.57

distance is: 7.87

distance is: 2.45

Circle test result:

Circle[center=Point: (0,0),radius=1.0]

Circle[center=Point: (1,2),radius=3.3]

Circle[center=Point: (4,5),radius=6.6]

Circle[center=Point: (11,12),radius=13.3]

center is: Point: (11,12)

radius is: 13.3

Circle[center=Point: (21,22),radius=13.3]

center's x is: 21

center's y is: 22

Circle[center=Point: (31,32),radius=13.3]

center's x is: 31

center's y is: 32

area is: 555.72

circumference is: 83.57

distance is: 7.87

distance is: 2.45

Result for Cylinder:

Radius is 1.0 Height is 1.0 Color is red Base area is 3.141592653589793 Volume is 3.141592653589793

Radius is 2.0 Height is 5.0 Color is red Base area is 12.566370614359172 Volume is 62.83185307179586

@overwrite notation: which asks compiler to check whether there is such a method in the superclass to be overridden. This helps greatly if you *misspell* the name of the method to be overridden

Inheritance:

the compressed code block⬆️

**class** Point{

**private** **float** x;

**private** **float** y;

**public** Point(){

}

**public** Point(**float** x, **float** y){

**this**.x = x;

**this**.y = y;

}

**public** **float** getX(){

**return** x;

}

**public** **void** setX(**float** x){

**this**.x = x;

}

**public** **float** getY(){

**return** y;

}

**public** **void** setY(**float** y){

**this**.y = y;

}

**public** **void** setXY(**float** x, **float** y){

**this**.x = x;

**this**.y = y;

}

**public** **float**[] getXY(){

**float**[] xy= {x, y};

**return** xy;

}

**public** String toString(){

**return** "(" + x + "," + y +")";

}

}

**class** MovablePoint **extends** Point{

**private** **float** xSpeed;

**private** **float** ySpeed;

MovablePoint(**float** x, **float** y, **float** xSpeed, **float** ySpeed){

**super**(x, y);

**this**.xSpeed = xSpeed;

**this**.ySpeed = ySpeed;

}

MovablePoint(){

}

**public** **float** getXSpeed(){

**return** xSpeed;

}

**public** **void** setXSpeed(**float** xSpeed){

**this**.xSpeed = xSpeed;

}

**public** **float** getYspeed(){

**return** ySpeed;

}

**public** **void** setYspeed(**float** ySpeed){

**this**.ySpeed = ySpeed;

}

**public** **void** setSpeed(**float** xSpeed, **float** ySpeed){

setXSpeed(xSpeed);

**this**.ySpeed = ySpeed;

}

**public** **float**[] getSpeed(){

**float**[] a = {xSpeed, ySpeed};

**return** a;

}

**public** String toString(){

**return** "(" + getX() + "," + getY() +"), speed = ("+ xSpeed + "," + ySpeed + ")";

}

**public** MovablePoint move(){

**float** x = getX();

**float** y = getY();

x += xSpeed;

y += ySpeed;

setX(x);

setY(y);

**return** **this**;

}

}



**public** **class** Point2D3D {

**public** **static** **void** main(String[] args){

Point2D a = **new** Point2D(3.4f, 3.5f);

System.out.println(a);

a.setXY(2F, 4F);

System.out.println(a);

Point2D b = **new** Point3D(3.5F, 3.7F, 3.6F);

System.out.println(b);

Point3D c = **new** Point3D();

System.out.println(c);

}

}

**class** Point2D{

**private** **float** x;

**private** **float** y;

**public** Point2D(){

}

**public** Point2D(**float** x, **float** y){

**this**.x = x;

**this**.y = y;

}

**public** **float** getX(){

**return** x;

}

**public** **void** setX(**float** x){

**this**.x = x;

}

**public** **float** getY(){

**return** y;

}

**public** **void** setY(**float** y){

**this**.y = y;

}

**public** **void** setXY(**float** x, **float** y){

**this**.x = x;

**this**.y = y;

}

**public** **float**[] getXY(){

**float**[] xy= {x, y};

**return** xy;

}

**public** String toString(){

**return** "(" + x + "," + y +")";

}

}

**class** Point3D **extends** Point2D{

**private** **float** z;

Point3D(){

**super**();

}

Point3D(**float** x, **float** y, **float** z){

**super**(x, y);

**this**.z = z;

}

**public** **float** getZ(){

**return** z;

}

**public** **void** setZ(**float** z){

**this**.z = z;

}

**public** **void** setXYZ(**float** x, **float** y, **float** z){

**super**.setXY(x, y);

setZ(z);

}

**public** **float**[] getXYZ(){

**float**[] xyz = {getX(), getY(), z};

**return** xyz;

}

**public** String toString(){

**return** "(" + getX() + "," + getY() + "," + z + ")";

}

}

Test result for point2D &3D:

(1,2)

(0,0)

(3,4)

x is: 3

x is: 4

(11,12,13)

(0,0,0)

(21,22,23)

x is: 21

y is: 22

z is: 23

Test result for person&Student&Teacher

Student: Tan Ah Teck(1 Happy Ave) IM101:97 IM102:68

Average is 82.5

Teacher: Paul Tan(8 sunset way)

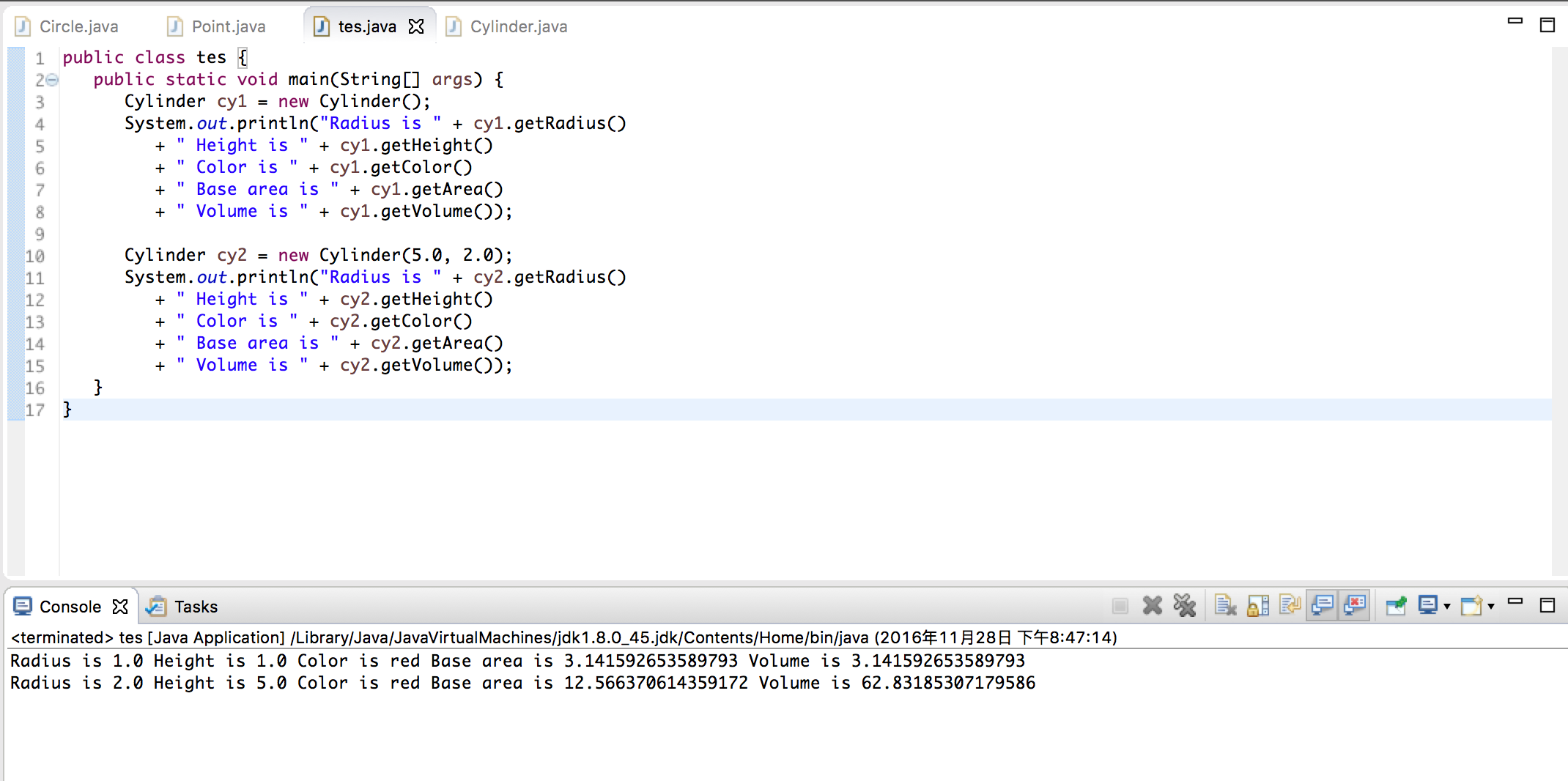
IM101 added.

IM102 added.

IM101 cannot be added.

IM101 removed.

IM102 removed.

IM101 cannot be removed.



Assigned Coursework:

2.1 cylinder:

**public** **class** Circle\_Cylinder {

**public** **static** **void** main (String[] args) {

*// Declare and allocate a new instance of cylinder*

*// with default color, radius, and height*

Cylinder c1 = **new** Cylinder();

System.out.println("Cylinder:"

+ " radius=" + c1.getRadius()

+ " height=" + c1.getHeight()

+ " base area=" + c1.getArea()

+ " volume=" + c1.getVolume());

*// Declare and allocate a new instance of cylinder*

*// specifying height, with default color and radius*

Cylinder c2 = **new** Cylinder(10.0);

System.out.println("Cylinder:"

+ " radius=" + c2.getRadius()

+ " height=" + c2.getHeight()

+ " base area=" + c2.getArea()

+ " volume=" + c2.getVolume());

*// Declare and allocate a new instance of cylinder*

*// specifying radius and height, with default color*

Cylinder c3 = **new** Cylinder(2.0, 10.0);

System.out.println("Cylinder:"

+ " radius=" + c3.getRadius()

+ " height=" + c3.getHeight()

+ " base area=" + c3.getArea()

+ " volume=" + c3.getVolume());

}

}

**class** Circle {

*// private instance variable*

**private** **double** radius;

*// Constructor*

**public** Circle(){

**this**.radius = 0;

}

**public** Circle(**double** radius) {

**this**.radius = radius;

}

*// Getter*

**public** **double** getRadius() {

**return** **this**.radius;

}

*// Return the area of this circle*

**public** **double** getArea() {

**return** radius \* radius \* Math.PI;

}

*// Describe itself*

**public** String toString() {

**return** "Circle[radius=" + radius + "]";

}

}

**class** Cylinder **extends** Circle { *// Save as "Cylinder.java"*

**private** **double** height; *// private variable*

*// Constructor with default color, radius and height*

**public** Cylinder() {

**super**(); *// call superclass no-arg constructor Circle()*

height = 1.0;

}

*// Constructor with default radius, color but given height*

**public** Cylinder(**double** height) {

**super**(); *// call superclass no-arg constructor Circle()*

**this**.height = height;

}

*// Constructor with default color, but given radius, height*

**public** Cylinder(**double** radius, **double** height) {

**super**(radius); *// call superclass constructor Circle(r)*

**this**.height = height;

}

*// A public method for retrieving the height*

**public** **double** getHeight() {

**return** height;

}

**public** **double** getArea(){

**return** 2 \* Math.PI \* getRadius() \* height + 2 \* getRadius() \* getRadius() \* Math.PI;

}

*// A public method for computing the volume of cylinder*

*// use superclass method getArea() to get the base area*

**public** **double** getVolume() {

**return** **super**.getArea()\*height;

}

**public** String toString() { *// in Cylinder class*

**return** "Cylinder: subclass of " + **super**.toString() *// use Circle's toString()*

+ " height=" + height;

}

}

4.1 Substitutability

With Substitutability we can create subclass instance and assign it to its superclass reference

You can invoke all the methods defined in the Circle class for the reference c1, (which is actually holding a Cylinder object), e.g.

However, you will get a compilation error invoke methods defined in the Cylinder class for the reference c1

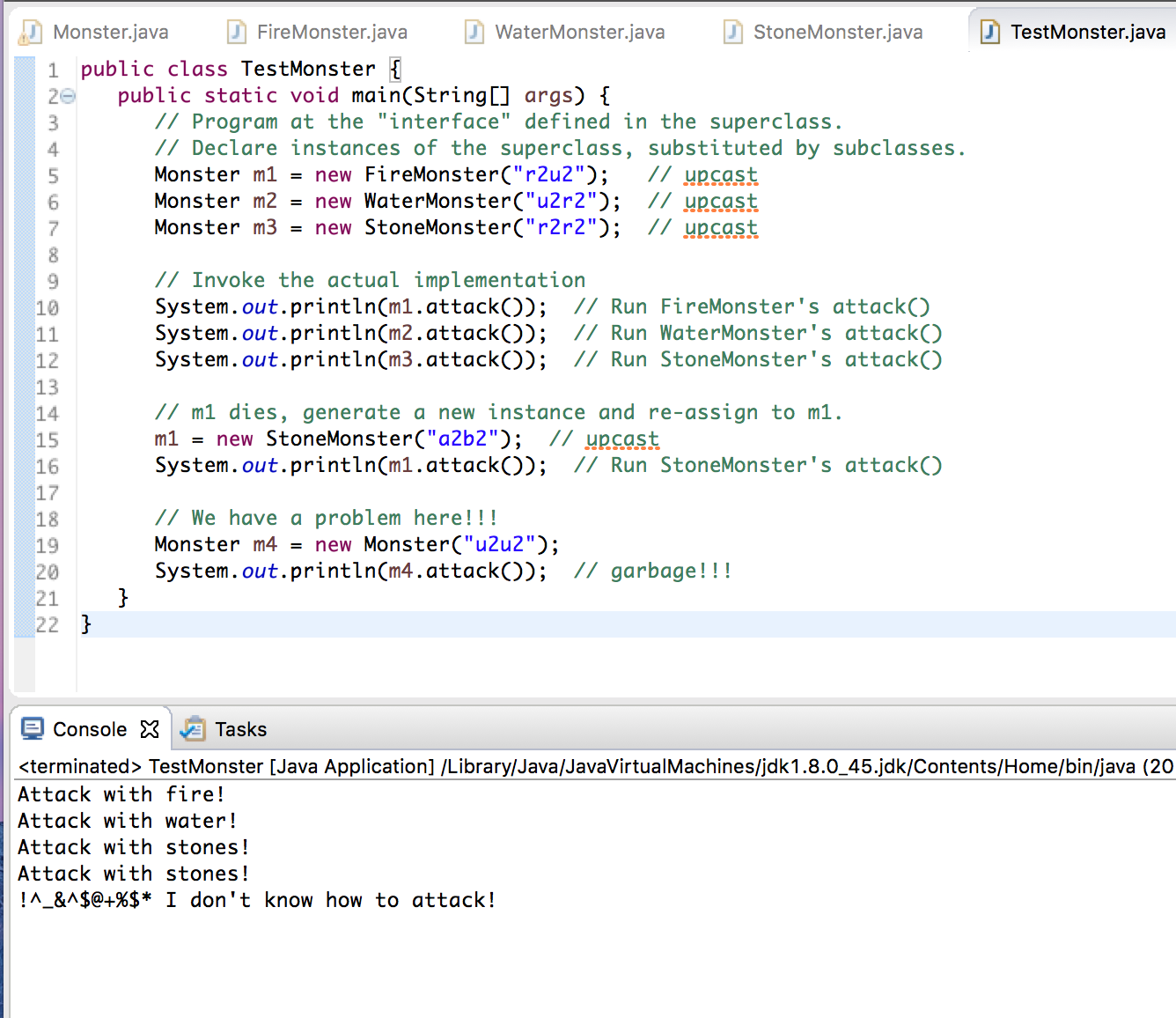
This is because c1 is a reference to the Circle class, which does not know about methods defined in the subclass Cylinder.

\*\*\*In a word: while declaring the instance reference determines the class it belongs in compilation phase

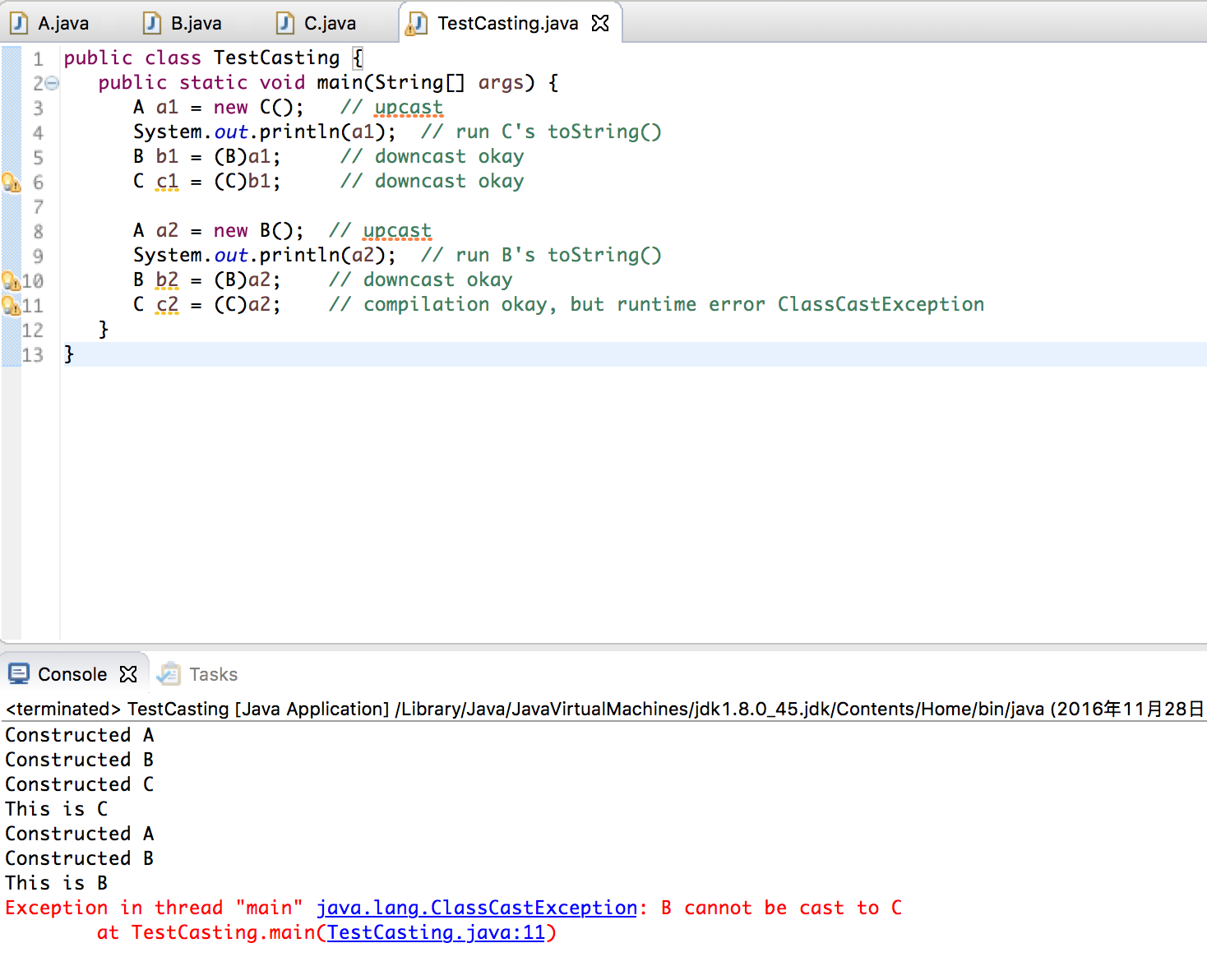
1 A subclass instance can be assigned (substituted) to a superclass' reference.

2 Once substituted, we can invoke methods defined in the superclass; we cannot invoke methods defined in the subclass.

3 However, if the subclass overrides inherited methods from the superclass, the subclass (overridden) versions will be invoked.\*\*\*

4.3  Polymorphism EG. 2： Monster and its Subclasses

This monster program gave us an intuitive sense of polymorphism in java. The method “attack” is identical in all of these three programs which extends the super class monster. The reference determines which method to call.

4.4 Upcasting & Downcasting

This code is example to show down-casting and upcasting.

Upcasting:Substituting a subclass instance for its superclass

Downcasting: Substituting a subclass instance for its superclass

The very last sentence throws an exception because the type of a2 is B, C can’t cast on B because it is not its subclass nor superclass

6.3 Interface

public interface Movable { // saved as "Movable.java"

public void moveUp();

public void moveDown();

public void moveLeft();

public void moveRight();

}

public class MovablePoint implements Movable { // saved as "MovablePoint.java"

// instance variables

int x, y, xSpeed, ySpeed; // package access

// Constructor

public MovablePoint(int x, int y, int xSpeed, int ySpeed) {

this.x = x;

}

// Implement abstract methods declared in the interface Movable

@Override

public void moveUp() {

y -= ySpeed; // y-axis pointing down for 2D graphics

}

public void movedown(){

y += ySpeed;

}

public void moveLeft(){

x += xSpeed;

}

public void moveRight(){

x -= xSpeed;

}

public abstract class MovableCircle implements Movable { // saved as "MovableCircle.java"

// instance variables

private MovablePoint center; // can use center.x, center.y directly

// because they are package accessible

private int radius;

// Constructor

public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {

// Call the MovablePoint's constructor to allocate the center instance.

center = new MovablePoint(x, y, xSpeed, ySpeed);

this.radius = radius;

}

// Implement abstract methods declared in the interface Movable

@Override

public void moveUp() {

center.y -= center.ySpeed; // y-axis pointing down for 2D graphics

}

public void movedown(){

center.y += center.ySpeed;

}

public void moveLeft(){

center.x -= center.xSpeed;

}

public void moveRight(){

center.x += center.xSpeed;

}

}

One characteristic about the interface is that it is like principle of the all class, every class must have the characteristics the interface specifies.