

# 제3장

# 상속과 Generic Programming

### 3.1 상속

# 상속 (Inheritance)

- A computer has  
manufacturer  
processor  
RAM  
disk  
processor speed



## Computer

```
String manufacturer
String processor
int ramSize
int diskSize
double processorSpeed
```

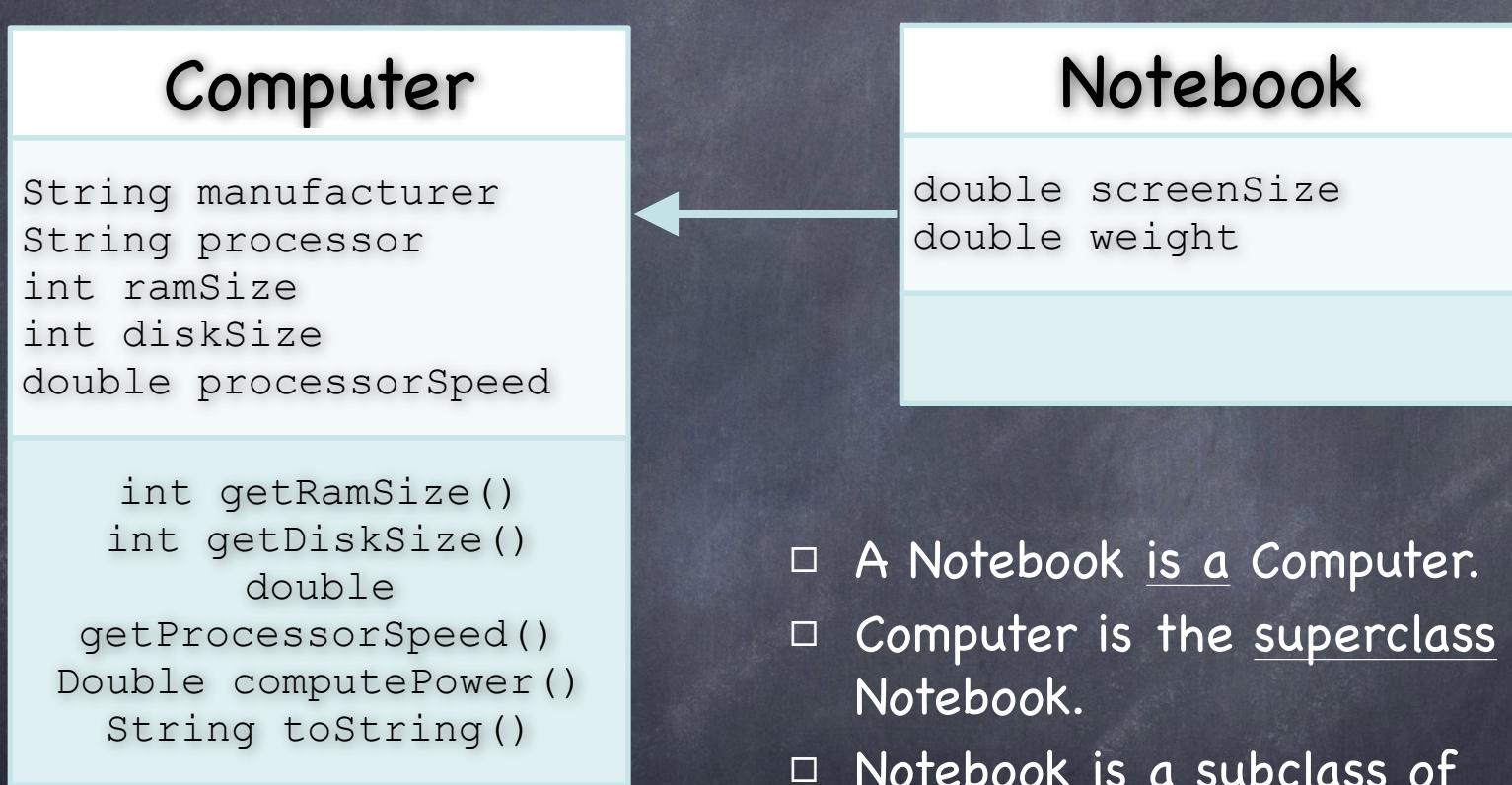
```
int getRamSize()
int getDiskSize()
double getProcessorSpeed()
Double computePower()
String toString()
```

# 상속 (Inheritance)

- ⦿ A Notebook has all the properties of Computer,
  - manufacturer
  - processor
  - RAM
  - Disk
  - processor speed
- ⦿ plus,
  - screen size
  - weight



# 상속 (Inheritance)



- A Notebook is a Computer.
- Computer is the superclass of Notebook.
- Notebook is a subclass of Computer.

# 상속 (Inheritance)

```
public class Computer {  
    private String manufacturer;  
    private String processor;  
    private int ramSize;  
    private int diskSize;  
    private double processorSpeed;  
  
    public Computer(String man, String proc, int ram, int disk, double procSpeed)  
    {  
        manufactuer = man;  
        this.processor = proc;  
        ramSize = ram;  
        diskSize = disk;  
        processorSpeed = procSpeed;  
    }  
}
```

# 상속 (Inheritance)

```
public double computePower() {  
    return ramSize * processorSpeed;  
}  
  
public double getRamSize() { return ramSize; }  
public double getProcessorSpeed() { return processorSpeed; }  
public int getDiskSize() { return diskSize; }  
  
public String toString() {  
    String result = "Manufacturer: " + manufacturer +  
        "\nCPU: " + processor +  
        "\nRAM: " + ramSize + " megabytes" +  
        "\nDisk: " + diskSize + " gigabytes" +  
        "\nProcessor speed: " + processorSpeed +  
        " gigahertz";  
  
    return result;  
}  
}
```

# 상속 (Inheritance)

```
public class Notebook extends Computer {  
    private double screenSize;  
    private double weight;  
    public Notebook(String man, String proc, int ram, int disk,  
                   double procSpeed, double screen, double weight) {  
        super(man, proc, ram, disk, procSpeed);  
        screenSize = screen;  
        this.weight = weight;  
    }  
}
```

## **super (arguments)**

수퍼클래스의 생성자 중에서 매개변수 리스트가 일치하는 생성자를 호출한다. **super ()**를 호출할 경우 반드시 생성자 내에서 첫 문장이어야 한다.

# 상속과 생성자

- 생성자가 없을 경우 자동으로 no-parameter 생성자가 만들어진다. 생성자가 하나라도 있을 경우 자동으로 만들어지지 않는다.
- 모든 서브 클래스의 생성자는 먼저 수퍼클래스의 생성자를 호출한다.
  1. `super(...)`를 통해 명시적으로 호출해 주거나,
  2. 그렇지 않을 경우에는 자동으로 no-parameter 생성자가 호출된다.
- 흔한 오류:  
수퍼클래스에 no-parameter 생성자가 없는데, 서브클래스의 생성자에서 `super(...)` 호출을 안해주는 경우

# Method Overriding

```
Computer myComputer =
```

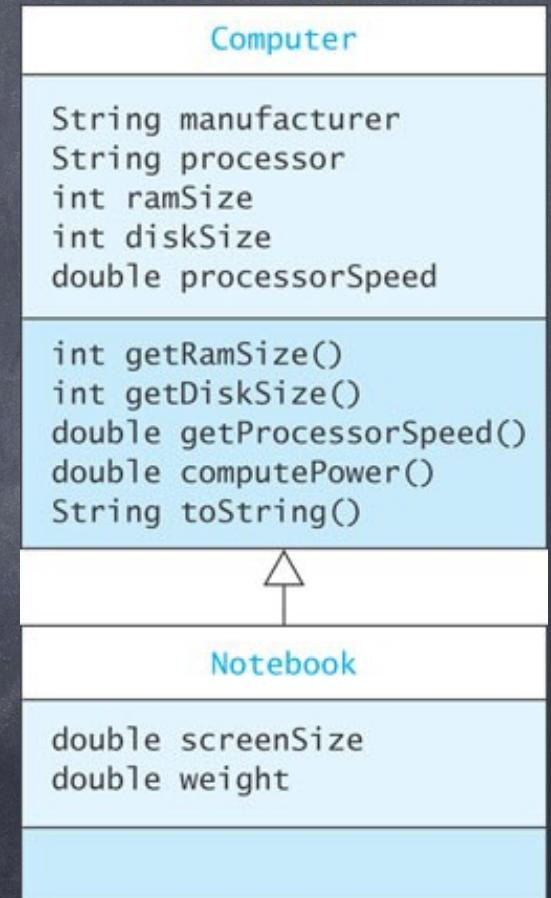
```
    new Computer("Acme", "Intel", 2, 160, 2.4);
```

```
Notebook yourComputer =
```

```
    new Notebook("DellGate", "AMD", 4, 240,  
                1.8, 15.0, 7.5);
```

```
System.out.println("My computer is:\n" +  
    myComputer.toString());
```

```
System.out.println("Your computer is:\n" +  
    yourComputer.toString());
```



# Method Overriding



출력:

My Computer is:

Manufacturer: Acme

CPU: Intel

RAM: 2.0 gigabytes

Disk: 160 gigabytes

Speed: 2.4 gigahertz

Your Computer is:

Manufacturer: DellGate

CPU: AMD

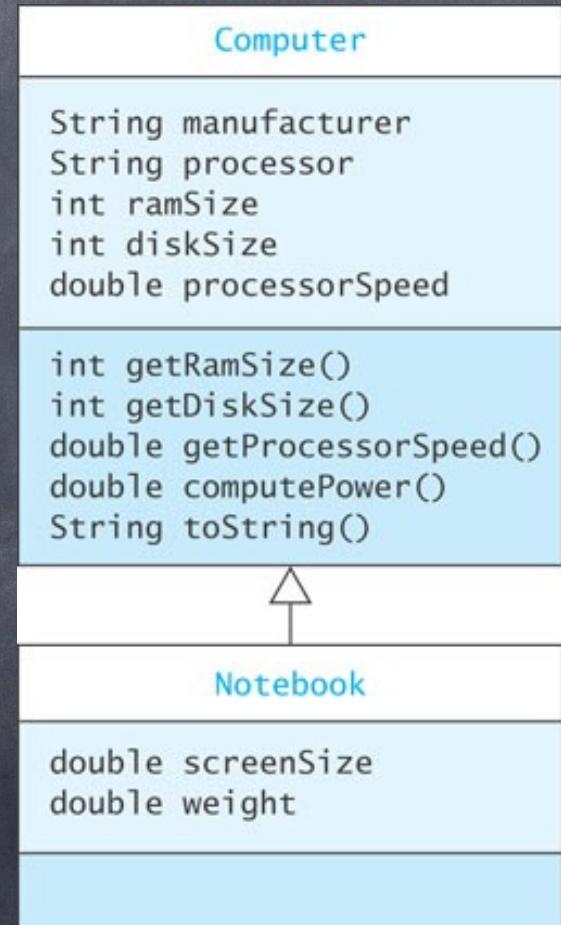
RAM: 4.0 gigabytes

Disk: 240 gigabytes

Speed: 1.8 gigahertz



screensize 와 weight 는 출력되지 않음



# Method Overriding

- 클래스 Notebook에 다음과 같이 `toString()` 메서드를 추가:

```
@Override  
public String toString() {  
    String result = super.toString() +  
        "\nScreen size: " + screenSize + " inches" +  
        "\nWeight: " + weight + " pounds";  
    return result;  
}
```

NoteBook의 super class 즉 Computer의  
`toString()` 메서드를 호출하였다.

- Notebook의 `toString()` 메서드가 Computer의 `toString()` 메서드를 `override` 함

# 다형성: Polymorphism

- 수퍼클래스 타입의 변수가 서브클래스 타입의 객체를 참조할 수 있다.

```
Computer theComputer = new Notebook("Bravo", "Intel", 4, 240, 2/4, 15.07.5);
```

Computer 타입의 변수 theComputer가  
Notebook 타입의 객체를 참조하고 있다.

```
System.out.println( theComputer.toString() );
```

theComputer는 Computer 타입의 변수이면서 실제로는 Notebook 개체를 참조하고 있다. 그리고 두 클래스는 각자의 `toString()`를 가지고 있다. 그렇다면 여기서 둘 중 어떤 `toString()` 메서드가 실행될까? Notebook 클래스의 `toString()` 메서드가 실행된다. 즉 동적 바인딩(dynamic binding)이 일어난다.

## 3.2 Case Study: 스케줄러 프로그램

# 4 종류의 이벤트

## ❶ 일회성 이벤트

생일, 식사약속, 회의 등

## ❷ 기간이 지정된 이벤트

시험기간, 축제기간 등

시작일과 종료일이 있는 경우

## ❸ 데드라인이 있는 이벤트

시작일은 없고 데드라인이 있는 일

제출기한이 있는 과제, 종료일이 있는 프로젝트 등

## ❹ 주기적 이벤트

수업시간 (3월 초에서 6월 말까지 매주 월, 목 등)

# 사용 예

\$ addevent oneday

when: 2017/1/15

title: K's birthday

\$ addevent duration

begin: 2017/1/10

end: 2017/1/31

title: Winter Festival

\$ addevent deadline

until: 2017/2/10

title: Submitting term paper

\$ addevent oneday

when: 2017/2/2

title: Project meeting

\$ list

K's birthday, 2017/1/15

Winter Festival, 2017/1/10 ~ 2017/1/31

Submitting term paper, 2017/2/10

Project meeting, 2017/2/2

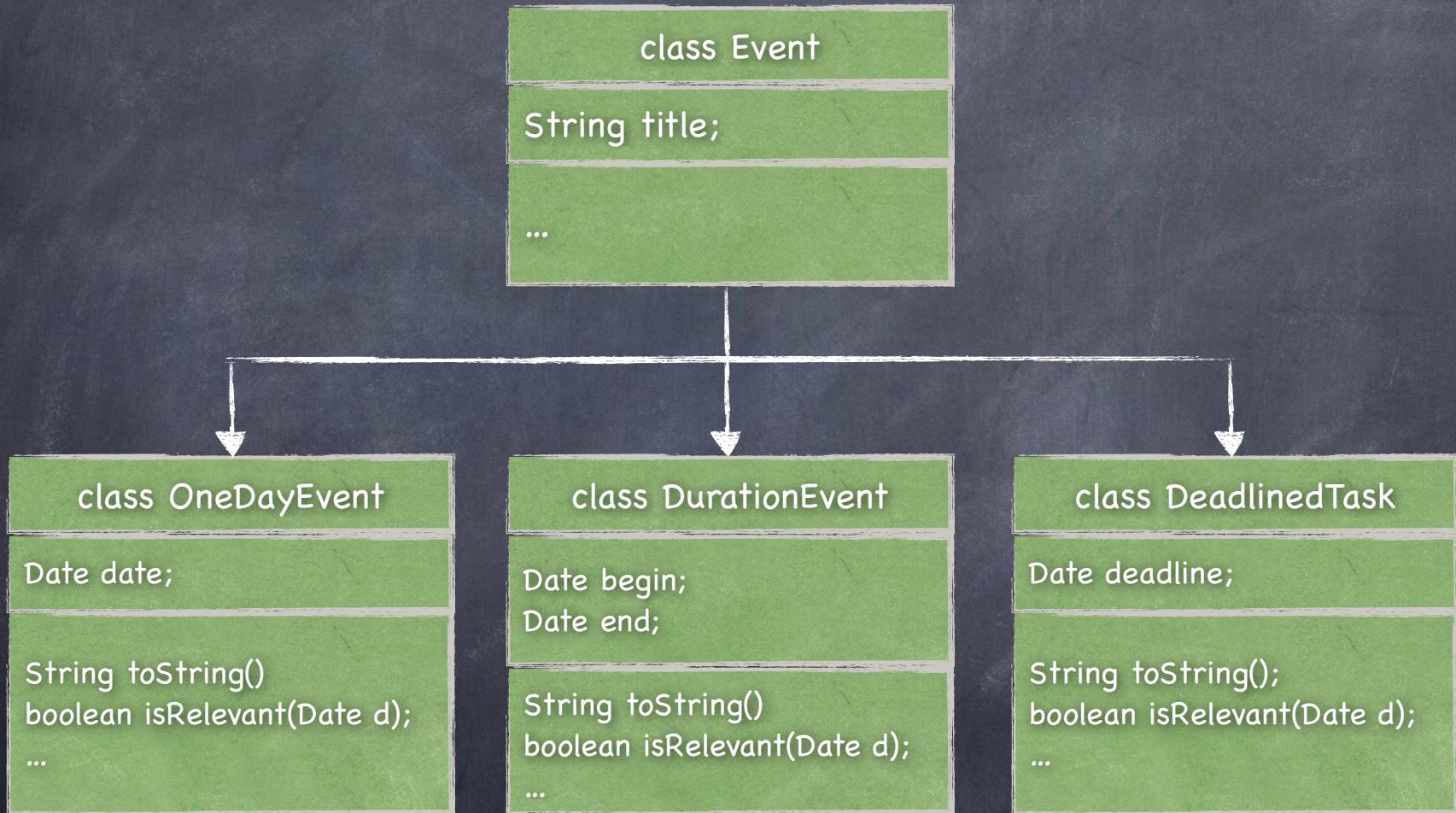
\$ show 2017/1/15

K's birthday, 2017/1/15

Winter Festival, 2017/1/10 ~ 2017/1/31

Submitting term paper, 2017/2/10

# Class Hierarchy



# class MyDate

```
public class MyDate {  
    public int year;  
    public int month;  
    public int day;  
  
    public MyDate(int y, int m, int d) {  
        year = y;    month = m;    day = d;  
    }  
  
    public String toString() {  
        return year + "/" + month + "/" + day;  
    }  
}
```

# class Event

```
public class Event {  
    public String title;  
    public Event(String title) {  
        this.title = title;  
    }  
}
```

# class OneDayEvent

```
public class OneDayEvent extends Event {  
    public MyDate date;  
    public OneDayEvent(String title, MyDate date)  
    {  
        super(title);  
        this.date = date;  
    }  
    public String toString()  
    {  
        return title + " ," + date.toString();  
    }  
}
```

# class DurationEvent

```
public class DurationEvent extends Event {  
    public MyDate begin;  
    public MyDate end;  
  
    public DurationEvent(String title, MyDate begin, MyDate end)  {  
        super(title);  
        this.begin = begin;  
        this.end = end;  
    }  
  
    public String toString()  {  
        return title + " ; " + begin.toString() + "~" + end.toString();  
    }  
}
```

# class DeadlinedTask

```
public class DeadlinedTask extends Event {  
    public MyDate deadline;  
    public DeadlinedTask(String title, MyDate date) {  
        super(title);  
        deadline = date;  
    }  
    public String toString() {  
        return title + ", " + deadline.toString();  
    }  
}
```

# class Scheduler

```
public class Scheduler {  
    public int capacity = 10;  
    public Event [] events;  
    public int n;  
    Scanner kb;  
  
    public Scheduler()  
    {  
        events = new Event [capacity];  
        n = 0;  
    }
```

# class Scheduler

```
public void processCommand()  {  
    kb = new Scanner( System.in );  
    while (true) {  
        System.out.print("$ ");  
        String command = kb.next();  
        if (command.equals("addevent")) {  
            String type = kb.next();  
            if (type.equalsIgnoreCase("OneDay"))  
                handleOneDayAdd( );  
            else if (type.equalsIgnoreCase("Duration"))  
                handleDurationAdd();  
            else if (type.equalsIgnoreCase("Deadline"))  
                handleDeadlineAdd();  
        }  
        else if (command.equals("list"))  
            handleList();  
        else if (command.equals("show"))  
            handleShow();  
        else if (command.equals("exit"))  
            break;  
    }  
    kb.close();  
}
```

# Splitting a String

```
String str = "012-3456-789";
String [] tokens = str.split("-");
for (int i=0; i<tokens.length; i++)
    System.out.println(tokens[i]);
```

str 012-3456-789

	0	1	2
tokens	012	3456	789

# Splitting a String

```
String str = "abc.defg.h.ijk";  
String [] tokens = str.split("\\.");  
for (int i=0; i<tokens.length; i++)  
    System.out.println(tokens[i]);
```

str abc.defg.h.ijk

tokens	0	1	2	3
	abc	def	h	ijk

# Splitting a String

"[ ]+" 대신 "\\s+"를 사용해도 된다. 이것을 정규표현식(regular expression)이라고 부른다: <http://docs.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html>

```
String [] tokens = str.split("[ ]+");
```

str [ This is a test string.]

	tokens	0	1	2	3	4
		This	is	a	test	string.

# class Scheduler

```
private MyDate parseDate(String dateString) {  
    String [] tokens = dateString.split("/");  
  
    int year = Integer.parseInt( tokens[0] );  
  
    int month = Integer.parseInt(tokens[1]);  
    int day = Integer.parseInt(tokens[2]);  
    MyDate date = new MyDate(year, month, day);  
    return date;  
}
```

예를 들어 문자열 "2017"에서  
정수 2017을 뽑아낸다.

# class Scheduler

```
private void handleOneDayAdd() {  
    System.out.print("  Title: ");  
    String title = kb.next();  
    System.out.print("  Date: ");  
    String dateString = kb.next();  
    OneDayEvent e = new OneDayEvent(title, parseDate(dateString));  
    addEvent(e);  
}
```

# class Scheduler

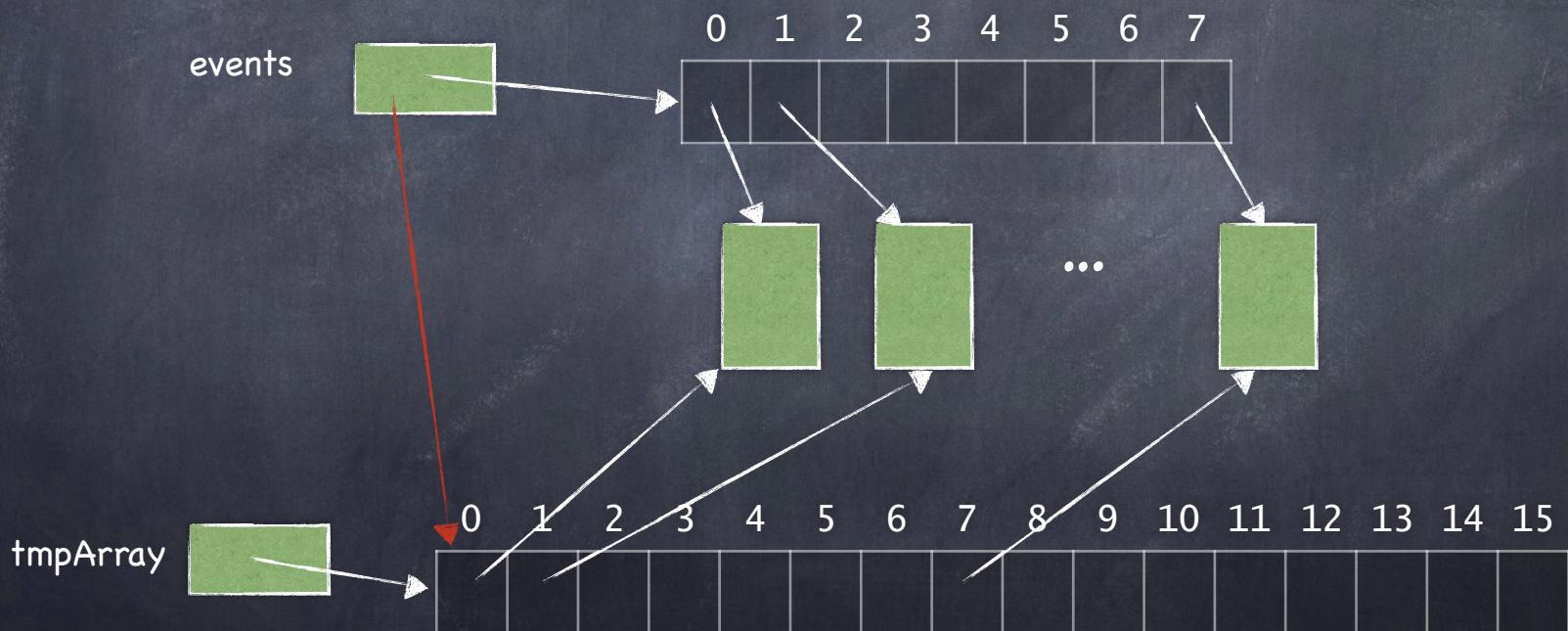
```
private void addEvent(Event ev) {  
    if ( n >= capacity )  
        reallocate();  
    events[n++] = ev;  
}
```

배열이 꽉찼다는 의미이다.

# 배열 재할당: Array Reallocation

```
private void reallocate() {  
    Event [] tmpArray = new Event [capacity * 2];  
    for (int i=0; i<n; i++)  
        tmpArray[i] = events[i];  
    events = tmpArray;  
    capacity *= 2;  
}
```

reallocate 메서드는 배열의 크기를 2배로 늘린다.



# class Scheduler

```
private void handleList() {  
    for (int i=0; i<n; i++)  
        System.out.println(" " + events[i].toString() );  
}
```

Dynamic binding이 일어나므로 각 이벤트 클래스의  
toString() 메서드가 호출된다.

# class Scheduler

```
private void handleShow() {  
    // 나중에...  
}  
  
private void handleDeadlineAdd() {  
    // 연습문제로 남겨둠  
}  
  
private void handleDurationAdd() {  
    // 연습문제로 남겨둠  
}  
  
public static void main(String [] args) {  
    Scheduler app = new Scheduler();  
    app.processCommand();  
}  
}
```

### 3.3 class Object와 Wrapper class

# class Object

- ☞ 클래스 Object는 Java에서 모든 클래스의 superclass이다.
- ☞ class Object의 멤버 메서드

Method	Behavior
<code>boolean equals(Object obj)</code>	Compares this object to its argument.
<code>int hashCode()</code>	Returns an integer hash code value for this object.
<code>String toString()</code>	Returns a string that textually represents the object.
<code>Class&lt;?&gt; getClass()</code>	Returns a unique object that identifies the class of this object.

- ☞ Java의 모든 클래스는 내가 만들어 주지 않아도 이미 equals와 toString 메서드를 가지고 있다.
- ☞ 다만 내 의도대로 작동하지는 않을 것이다.

# `toString()`

- ☞ Java의 모든 클래스는 내가 만들어 주지 않아도 이미 `equals`와 `toString` 메서드를 가지고 있다. 다만 내 의도대로 작동하지는 않을 것이다.
- ☞ 만약 `toString` 메서드를 따로 만들어주지 않은 클래스의 객체에 대해서 `toString()` 메서드를 호출하면 다음과 같은 `String`이 반환된다.

예: `PhoneBook@ef08879` (클래스 이름@객체의 hash code)

# equals(Object)

- Object 클래스의 equals 메서드의 매개변수는 Object 타입이다.

```
public boolean equals (Object other) { ... }
```

- 매개변수로 제공된 객체와 자기 자신의 동일성을 검사한다.
- 이 메서드를 의도대로 사용하려면 override해야 한다.

# equals() 메서드 오버라이딩 하기

@Override

```
public boolean equals(Object obj) {  
    if (obj == this) return true;  
    if (obj == null) return false;  
    if (this.getClass() == obj.getClass()) {  
        Person other = (Person) obj;  
        return name.equals(other.name) && number.equals(other.number);  
    } else {  
        return false;  
    }  
}
```

class Person에서 equals 메서드를  
override한 예이다.

두 Person 객체가 name이 동일하면 동일하다고 할  
것인지 number까지 동일해야 동일하다고 할지는  
내가 결정할 문제이다.

# Class Class

- 모든 클래스는 하나의 *Class* 객체를 가진다.
- 이 객체는 각각의 클래스에 대해서 유일(unique)하다.
- 메서드 `getClass()`는 *Object* 클래스가 제공하는 메서드이며, 이 유일한 *Class* 객체 반환한다.
- 앞 페이지의 예에서 만약 `this.getClass() == obj.getClass()` 가 `true`라면 우리는 비교 대상인 두 객체 (`this`와 `obj`)가 동일한 클래스의 객체임을 알 수 있다.

# Wrapper class

- Java에서 primitive type 데이터와 non-primitive type 데이터, 즉 객체는 근본적으로 다르게 처리된다.
- 가령 Object 타입의 배열에는 다형성의 원리에 의해서 모든 종류의 객체를 저장할 수 있다. 하지만, int, double, char 등의 primitive type 데이터는 저장할 수 없다. 객체가 아니므로...
- 때때로 primitive type 데이터를 객체로 만들어야 할 경우가 있다. 이럴 때 Integer, Double, Character 등의 wrapper class를 이용한다.

# Wrapper class

- 기본 타입의 데이터를 하나의 객체로 포장해주는 클래스:
- *Integer, Double, Character, Boolean* 등

```
int a = 20;  
Integer age = new Integer(a);  
int b = age.intValue();           // b becomes 20
```

- 데이터 타입간의 변환 기능을 제공

```
String str = "1234";  
int d = Integer.parseInt(str);    // d becomes 1234
```

# Wrapper class

```
Object [] theArray = new Object [100];  
theArray[0] = new Integer(10);  
theArray[1] = new Person("John", "10101010");  
  
Integer value = (Integer) theArray[0];  
  
int a = value.intValue();
```

비록 실제로 theArray[0]에 저장된 것은 Integer 타입일지라고 이렇게 타입변환을 해주지 않으면 컴파일러가 오류로 판정한다.

# Autoboxing과 Unboxing

```
Object [] theArray = new Object [100];  
theArray[0] = 10;
```

```
int a = (Integer)theArray[0];
```

theArray[0]에 저장된 것은 Integer 객체이지만 Java 컴파일러가  
자동으로 정수로 변환해준다. 이것을 unboxing이라고 부른다.

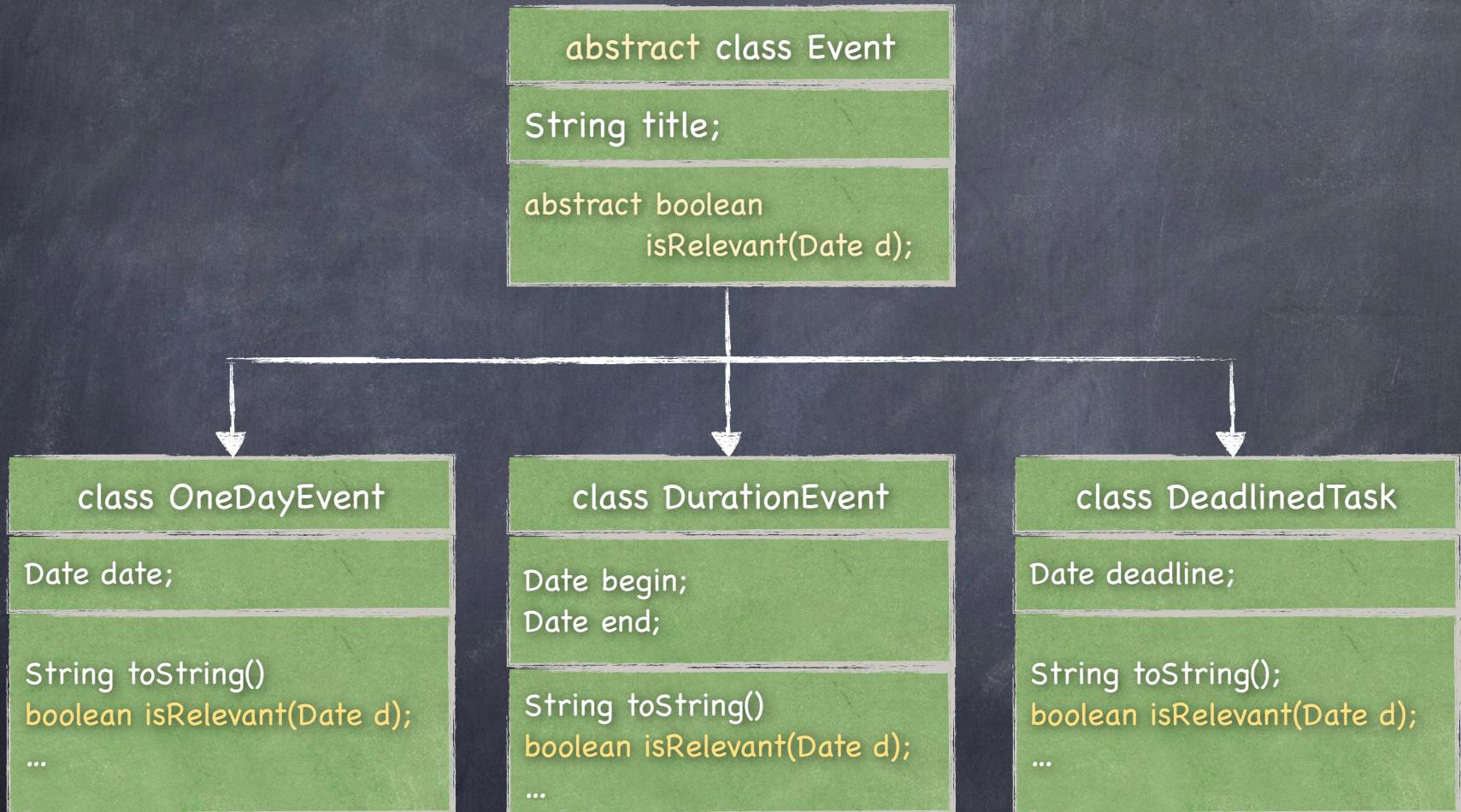
10은 정수이다. 하지만 이 경우 Java  
컴파일러가 자동으로 이것을 Integer  
객체로 변환해준다. 이것을 **autoboxing**  
이라고 부른다.

## 3.4 추상 클래스와 인터페이스

# 추상 클래스

- 추상(abstract) 메서드는 선언만 있고 구현이 없는 메서드
- 추상 메서드를 포함한 클래스는 추상 클래스
- 추상 메서드와 추상 클래스는 키워드 `abstract`로 표시
- 추상 클래스는 객체를 만들 수 없음. 따라서 서브 클래스를 만드는 용도로만 사용됨

# Class Hierarchy



# class Event

```
public abstract class Event {  
    public String title;  
    public Event(String title) {  
        this.title = title;  
    }  
    public abstract boolean isRelevant(MyDate date);  
}
```

# class MyDate

```
public class MyDate {  
    public int year;  
    public int month;  
    public int day;  
  
    public MyDate(int y, int m, int d) {  
        year = y; month = m; day = d;  
    }  
  
    public int compareTo( MyDate other ) {  
        if ( year < other.year || year == other.year && month < other.month ||  
            year == other.year && month == other.month && day < other.day)  
            return -1;  
        else if ( year > other.year || year == other.year && month > other.month ||  
            year == other.year && month == other.month && day > other.day)  
            return 1;  
        else  
            return 0;  
    }  
  
    public String toString() {  
        return year + "/" + month + "/" + day;  
    }  
}
```

# class OneDayEvent

```
public class OneDayEvent extends Event {  
    public MyDate date;  
    public OneDayEvent(String title, MyDate date) {  
        super(title);  
        this.date = date;  
    }  
  
    public boolean isRelevant(MyDate theDay) {  
        return date.compareTo(theDay) == 0;  
    }  
  
    public String toString()  
    {  
        return title + ", " + date.toString();  
    }  
}
```

# class DurationEvent

```
public class DurationEvent extends Event {  
    public MyDate begin;  
    public MyDate end;  
  
    public DurationEvent(String title, MyDate begin, MyDate end) {  
        super(title);  
        this.begin = begin;  
        this.end = end;  
    }  
  
    public boolean isRelevant(MyDate date) {  
        return begin.compareTo(date) <= 0 && end.compareTo(date) >= 0;  
    }  
  
    public String toString() {  
        return title + ", " + begin.toString() + "~" + end.toString();  
    }  
}
```

# class DeadlinedTask

```
public class DeadlinedTask extends Event {  
    public MyDate deadline;  
    public DeadlinedTask(String title, MyDate date) {  
        super(title);  
        deadline = date;  
    }  
    public boolean isRelevant(MyDate date) {  
        return date.compareTo(deadline) <= 0;  
    }  
    public String toString() {  
        return title + " , " + deadline.toString();  
    }  
}
```

# class Scheduler

```
private void handleShow() {  
    String dateString = kb.next();  
    MyDate theDate = parseDateString(dateString);  
    for (int i=0; i<n; i++)  
        if ( events[i].isRelevant( theDate ) )  
            System.out.println( events[i].toString() );  
}
```

# 인터페이스

## 眼球 인터페이스

추상 메서드만을 가진 순수한 추상 클래스

static final 데이터 멤버 (상수)를 가질 수 있음

## 眼球 예:

```
public interface Payable {  
    public double calcSalary();  
    public boolean salaried();  
    public static final double DEDUCTIONS = 25.5;  
}
```

```
public class Professor implements Payable {
```

...

```
    public boolean calcSalary() { .... }
```

```
    public boolean salaried() { .... }
```

...

```
}
```

Payable 인터페이스를 구현(implements)하는  
클래스는 메서드 calcSalary와 salaried를  
실제로 구현해야 한다.

# Shape 프로그램

- ▣ 사각형, 원, 직각삼각형 등의 도형들을 입력받아 저장하고
- ▣ 면적과 둘레 길이를 계산하는 기능

# 실행 예

```
$ add R 1 2
```

```
$ add C 5
```

```
$ show
```

1. Rectangle: width is 1, height is 2

2. Circle: radius is 5

```
$ showdetail
```

1. Rectangle: width is 1, height is 2

- The area is 2.0

- The perimeter is 6.0

2. Circle: radius is 5

- The area is 78.53981633974483

- The perimeter is 31.41592653589793

# 실행 예 (계속)

```
$ add R 2 6
```

```
$ sort          // 면적에 대해서 오름차순으로 정렬한다.
```

```
$ showdetail
```

1. Rectangle: width is 1, height is 2

- The area is 2.0
- The perimeter is 6.0

2. Rectangle: width is 2, height is 6

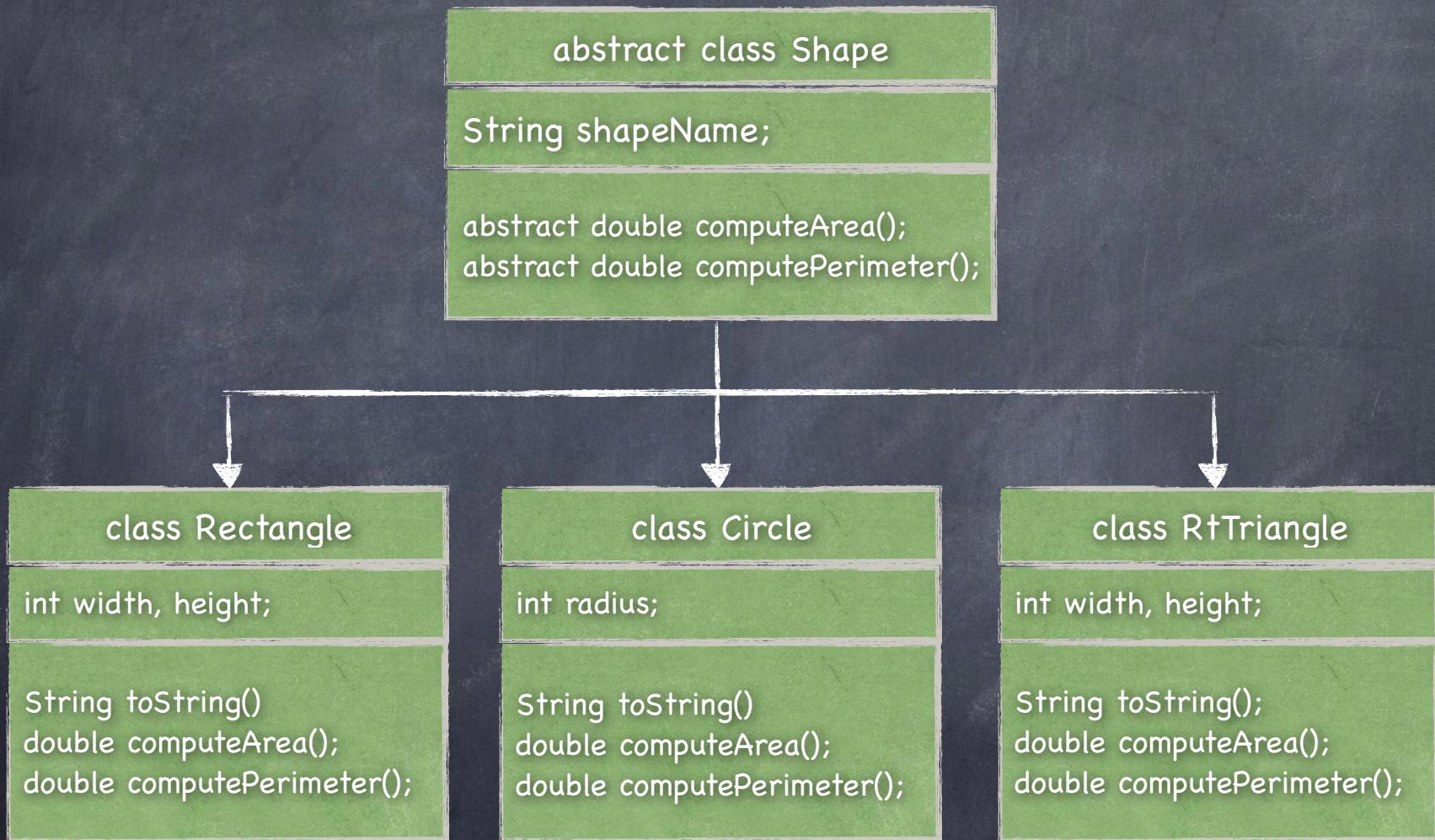
- The area is 12.0
- The perimeter is 16.0

3. Circle: radius is 5

- The area is 78.53981633974483
- The perimeter is 31.41592653589793

```
$ exit
```

# Class Hierarchy



```
public abstract class Shape {  
    private String shapeName;  
    public Shape(String name) {  
        shapeName = name;  
    }  
    public abstract double computeArea();  
    public abstract double computePerimeter();  
}
```

```
public class Rectangle extends Shape {  
    private int width = 0;  
    private int height = 0;  
    public Rectangle(int w, int h) {  
        super("Rectangle");  
        width = w;  
        height = h;  
    }  
    public int getWidth() {  
        return width;  
    }  
    public int getHeight() {  
        return height;  
    }  
}
```

```
public double computeArea() {  
    return (double)height * width;  
}  
  
public double computePerimeter() {  
    return 2.0 * (height + width);  
}  
  
public String toString() {  
    return "Rectangle: width is " + width + ", height is " + height;  
}  
}
```

```
public class Circle extends Shape {  
    private int radius;  
    public Circle( int rad ) {  
        super("Circle");  
        radius = rad;  
    }  
    public void setRadius( int rad ) {  
        radius = rad;  
    }  
    public int getRadius() {  
        return radius;  
    }  
    public double computeArea() {  
        return radius * radius * Math.PI;  
    }  
    public double computePerimeter() {  
        return 2 * radius * Math.PI;  
    }  
    public String toString() {  
        return "Circle radius is " + radius;  
    }  
}
```

```
public class RtTriangle extends Shape {
```

```
/* omitted */
```

```
}
```

```
public class ShapeApplication {  
    private static final int INITIAL_CAPACITY = 100;  
    private int capacity = INITIAL_CAPACITY;  
    private int n = 0;  
    private Shape [] shapes = new Shape [capacity];  
    private Scanner kb = new Scanner(System.in);  
  
    public static void main(String [] args) {  
        ShapeApplication app = new ShapeApplication();  
        app.processCommand();  
    }  
}
```

```
public void processCommand() {  
    while(true) {  
        System.out.print("$ ");  
        String command = kb.next();  
        if (command.equals("add"))  
            handleAdd();  
        else if (command.equals("show") || command.equals("showdetail"))  
            handleShow( command.equals("showdetail") );  
        else if (command.equals("sort"))  
            bubbleSort();  
        else if (command.equals("exit"))  
            break;  
    }  
}
```

```
public void handleAdd()
{
    String type = kb.nextInt();
    switch (type) {
        case "R":
            addShape(new Rectangle(kb.nextInt(), kb.nextInt()));
            break;
        case "C":
            addShape(new Circle(kb.nextInt()));
            break;
        case "T":
            addShape(new RtTriangle(kb.nextInt(), kb.nextInt()));
            break;
    }
}
```

```
public void addShape(Shape shape) {  
    if (n >= capacity)  
        reallocate();  
    shapes[n++] = shape;  
}  
  
private void reallocate() {  
    capacity *= 2;  
    Shape[] tmp = new Shape[capacity];  
    System.arraycopy(shapes, 0, tmp, 0, shapes.length);  
    shapes = tmp;  
}
```

```
public void handleShow(boolean detailed) {  
    for (int i=0; i<n; i++) {  
        System.out.println(" " + i + ". " + shapes[i].toString());  
        if ( detailed ) {  
            System.out.println(" - The area is " + shapes[i].computeArea());  
            System.out.println(" - The perimeter is " +  
                shapes[i].computePerimeter());  
        }  
    }  
}
```

# 삽입정렬

```
public void bubbleSort() {  
    for ( int i=n-1; i>0; i— ) {  
        for (int j=0; j<i; j++) {  
            if ( shapes[j].computeArea() > shapes[j+1].computeArea() ) {  
                Shape tmp = shapes[j];  
                shapes[j] = shapes[j+1];  
                shapes[j+1] = tmp;  
            }  
        }  
    }  
}
```

이 정렬 메서드는 오직 Shape 데이터들을 정렬하는  
용도로만 사용가능하다. 즉 generic하지 않다.

# *Comparable* 인터페이스

```
public interface Comparable {  
    int compareTo(Object o);  
}
```

*Comparable* 인터페이스는 Java API에 이미 정의되어 있으므로 우리가 새로 정의할 필요는 없다.

# 삽입정렬

```
public void sort( Comparable [] data, int size )
{
    for ( int i=size-1; i>0; i-- ) {
        for (int j=0; j<i; j++) {
            if ( data[j].compareTo(data[j+1]) > 0 ) {
                Comparable tmp = data[j];
                data[j] = data[j+1];
                data[j+1] = tmp;
            }
        }
    }
}
```

이 정렬 메서드는 Comparable 인터페이스를 구현하는 어떤 클래스의 객체이든 정렬할 수 있다. 즉 더 generic하다. 이렇게 generic한 코드는 재 사용의 측면에서 훨씬 유리하다.

```
public abstract class Shape implements Comparable
{
    private String shapeName;
    public Shape(String name) {
        shapeName = name;
    }
    public abstract double computeArea();
    public abstract double computePerimeter();

    public int compareTo(Object obj) {
        double mine = this.computeArea();
        double yours = ((Shape)obj).computeArea();
        if (mine < yours)
            return -1;
        else if (mine==yours)
            return 0;
        else
            return 1;
    }
}
```

```
public void processCommand() {  
    while(true) {  
        System.out.print("$ ");  
        String command = kb.next();  
        if (command.equals("add"))  
            handleAdd();  
        else if (command.equals("show") || command.equals("showdetail"))  
            handleShow( command.equals("showdetail") );  
        else if (command.equals("sort"))  
            Arrays.sort( shapes, 0, n );  
        else if (command.equals("exit"))  
            break;  
    }  
}
```

실제로는 정렬 메서드를 따로 구현할 필요도 없이  
Java의 `Arrays` 클래스가 제공하는 `sort` 메서드를  
이용하여 정렬할 수 있다.

# Interface vs. Abstract Class

- 추상 메서드로만 구성된 추상 클래스는 인터페이스와 완전히 동일한가?
- 다중 상속(multiple inheritance)

Java에서는 다중 상속을 허용하지 않는다.

하지만 하나의 클래스가 여러 개의 Interface를 implement하는 것은 가능

## 3.5 Generic 프로그래밍과 *Generics*

# Generic Programming

- ❶ 제네릭 프로그래밍 (Generic programming)은 데이터 형식에 의존하지 않고, 하나의 값이 여러 다른 데이터 타입들을 가질 수 있는 기술에 중점을 두어 재사용성을 높일 수 있는 프로그래밍 방식이다. [위키백과]
- ❷ “Generic programming is a style of computer programming in which algorithms are written in terms of types to-be-specified-later that are then instantiated when needed for specific types provided as parameters.” [Wikipedia]

# Generic Programming

- ⦿ Generic한 변수/자료구조

Event ev;

Event [] events = new Event [capacity];

Object obj;

- ⦿ Generic한 알고리즘 (method)

Arrays.sort( shapes, 0, n );

- ⦿ Generic 클래스

Generics

# Generics

```
public class Box<T> {  
    private T t;  
    public void set(T t) { this.t = t; }  
    public T get() { return t; }  
}
```

T라는 가상의 타입에 의해서  
parameterized된 클래스

...  
객체를 생성하는 시점에 가상의 타입 T를 실제  
하는 타입으로 지정해준다.

```
Box<Integer> integerBox = new Box<Integer>();  
integerBox.set( new Integer(10) );  
Box<Event> eventBox = new Box<Event>();  
eventBox.set(new OneDayEvent("dinner", new MyDate(2017,2,10)));
```

# Generics

```
public class Pair<K, V> {  
    private K key;  
    private V value;  
    public void set(K key, V value) { this.key = key; this.value = value}  
    public K getKey() { return key; }  
}
```

2개 이상의 type parameter를 가질  
수도 있다.

...  
객체를 생성하는 시점에 가상의 타입 K와 V를  
실제하는 타입으로 지정해준다.

```
Pair<String, Integer> p1 = new Pair<String, Integer>();  
p1.set("Even", 9);  
Pair<String, String> p2 = new Pair<String, String>();
```

# class Object vs. Generics

```
public class Box {  
    private Object t;  
    public void set(Object t) { this.t = t; }  
    public Object get() { return t; }  
}
```

```
Box box = new Box();  
box.set( new Integer(10) );  
Integer a = (Integer)box.get();
```

```
public class Box<T> {  
    private T t;  
    public void set(T t) { this.t = t; }  
    public T get() { return t; }  
}
```

```
Box<Integer> box = new Box<Integer>();  
box.set( new Integer(10) );  
Integer a = box.get();
```

get()의 return type은 Object이므로 반드시  
type casting을 해주어야 한다.

# Generic한 리스트 클래스를 만들어보자

## ☞ 리스트 (list)

여러 개의 데이터를 저장하고,  
임의의 위치에 새로운 데이터를 추가하거나  
데이터의 삭제가 가능하고  
임의의 위치의 데이터를 읽을 수 있고,  
용량에 제한이 없고,

...

# 사용 예

- 예를 들어 *String*들을 저장하려면

```
MyArrayList<String> myList = new MyArrayList<String>();
```

- 생성된 리스트에 *String* 추가

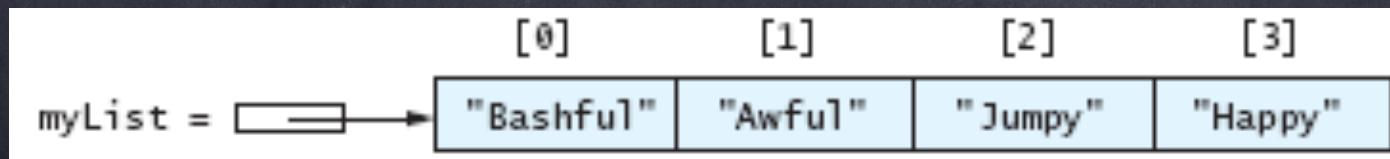
```
myList.add("Bashful");
```

```
myList.add("Awful");
```

```
myList.add("Jumpy");
```

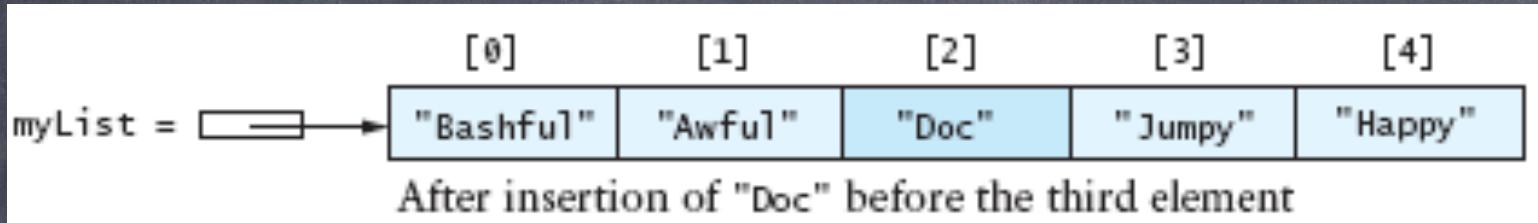
```
myList.add("Happy");
```

- 추가한 순서대로 저장됨

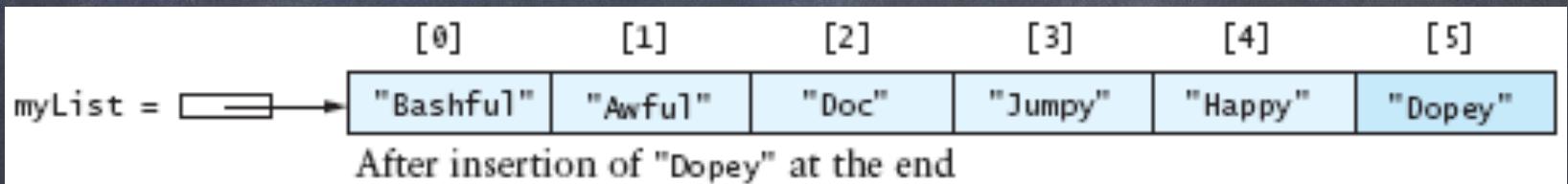


# 사용 예

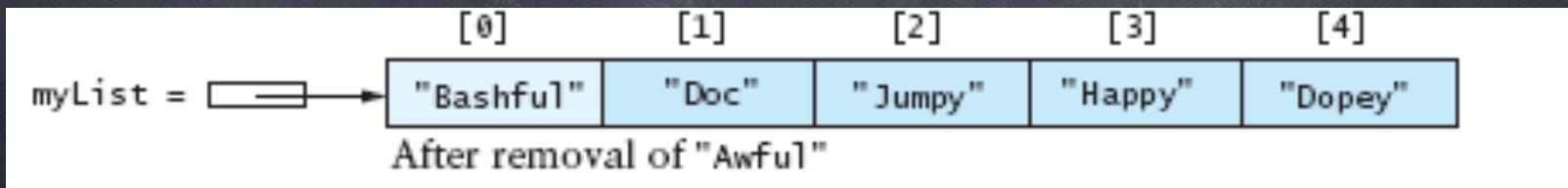
④ myList.add(2, "Doc");



④ myList.add("Dopey");

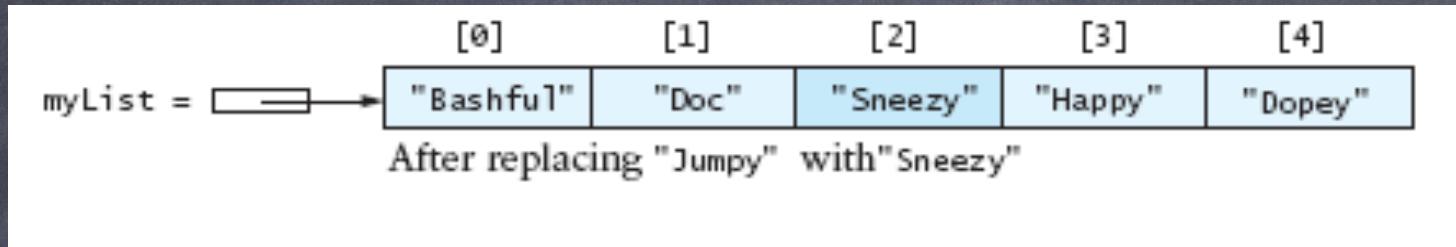


④ myList.remove(1);



# 사용 예

- myList.set(2, "Sneezy");



- String dwarf = myList.get(2);

dwarf의 값은?

- int index = myList.indexOf("Sneezy");

index의 값은?

- int index2 = myList.indexOf("Jumpy");

index2의 값은?

# MyArrayList

```
public class MyArrayList<E> {  
    private static final int INITIAL_CAPACITY = 10;  
    private E [] theData;  
    private int size = 0;  
    private int capacity = 0;
```

# MyArrayList

```
public MyArrayList () {  
    capacity = INITIAL_CAPACITY;  
    theData = (E []) new Object[capacity];  
}
```

This statement allocates storage for an array of type Object and then casts the array object to type E[]  
Although this may cause a compiler warning, it's ok

# add(index,E)

```
public void add (int index, E anEntry) {  
    if (index < 0 || index > size) {  
        throw new ArrayIndexOutOfBoundsException(index);  
    }  
    if (size >= capacity) {  
        reallocate();  
    }  
    for (int i = size; i > index; i--) {  
        theData[i] = theData[i-1];  
    }  
    theData[index] = anEntry;  
    size++;  
}
```

# size, add, indexOf

```
public int size()  
{  
    return size;  
}
```

```
public void add(E anEntry) {  
    add(size(), anEntry);  
}
```

```
public int indexOf(E anEntry) {  
    for (int i=0; i<size; i++)  
        if (theData[i].equals(anEntry))  
            return i;  
    return -1;  
}
```

indexOf 메서드가 의도한 대로 작동 하려면 클래스 E는 equals 메서드를 overriding해야 한다.

# set, get

```
public E get (int index) {  
    if (index < 0 || index >= size) {  
        throw new ArrayIndexOutOfBoundsException(index);  
    }  
    return theData[index];  
}
```

```
public E set (int index, E newValue) {  
    if (index < 0 || index >= size) {  
        throw new ArrayIndexOutOfBoundsException(index);  
    }  
    E oldValue = theData[index];  
    theData[index] = newValue;  
    return oldValue;  
}
```

# remove

```
public E remove (int index) {  
    if (index < 0 || index >= size) {  
        throw new ArrayIndexOutOfBoundsException(index);  
    }  
    E returnValue = theData[index];  
    for (int i = index + 1; i < size; i++) {  
        theData[i-1] = theData[i];  
    }  
    size--;  
    return returnValue;  
}
```

# realloc

```
private void reallocate () {  
    capacity *= 2;  
    theData = Arrays.copyOf(theData, capacity);  
}
```

# MyArrayList의 활용

- Scheduler프로그램을 MyArrayList를 사용하도록 수정해보자.

# Vector와 ArrayList

- Java API `java.util`은 `Vector`와 `ArrayList`라는 두 가지 유사한 기능의 클래스를 제공
- 우리가 작성한 `MyArrayList`는 `ArrayList`의 축소판
- `Vector` 보다 `ArrayList`가 좀 더 효율적이며 주로 사용됨
- `ArrayList`와는 달리 `Vector` 클래스는 `synchronized`됨, 즉 다수의 `thread`가 중 돌없이 `Vector` 객체를 액세스 할 수 있음
- `CopyOnWriteArrayList` 클래스