به نام پروردگار دانایی

طراحی سیستم‌های شی‌گرا

برنامه‌نویسی شی‌گرا

درس سوم: تعامل بین اشیا

سید کاوه احمدی

گرم کردن!

- کلاسی برای تنظیم دما بنویسید به نحوی که بتواند دمای محیط را و حداکثر دمای مجاز را نگهداری کند. دمای اولیه محیط برابر صفر است و حداکثر دمای مجاز توسط کاربر مشخص می‌شود. می‌توانیم دمای محیط را افزایش دهیم. میزان افزایش را کاربر مشخص می‌کند. اگر دمای فعلی محیط از حداکثر دمای مجاز مشخص شده بیشتر شد، دما افزایش نیابد و یک پیغام خطا چاپ کند.

این جلسه

- تکمیل مباحث جلسه قبل
  - مفاهیم و تکنیک‌های طراحی: تجرید و واحدبندی
  - دیاگرام شی برای نمایش وضعیت برنامه در حال اجرا
  - مباحث بیشتر پیرامون ایجاد شی و فراخوانی متدها
  - تعامل

ساعت دیجیتال

- می‌خواهیم یک ساعت دیجیتال پیاده سازی کنیم.

تجرید و واحدبندی (Abstraction and modularization)

- تجرید
  - نادیده گرفتن جزئیات برای متمرکز شدن توجه بر سطح بالاتری از یک مسئله.
  - رفتار کردن هر واحد به شکل یک واحد اتومیک یا "جعبه سیاه" (black box).
    - نمی‌دانم موتور ماشین چگونه کار می‌کند اما می‌توانم رانندگی کنم.
    - برای من، ماشین یک "جعبه سیاه" است.
- واحدبندی
  - پروسه تقسیم‌بندی یک مسئله به قسمت‌های خوش تعریف است به نحوی که بتوانند به طور جداگانه تولید و آزمایش شوند و به روش‌های خوش تعریفی بتوانند تعامل برقرار کنند.

واحدبندی

می‌خواهید برای تعطیلات به اسپانیا بروید

- می‌توانید مسئله را به چند واحد تقسیم کنید:
  - رفتن به فرودگاه
  - پرواز به اسپانیا
  - رفتن از فرودگاه در اسپانیا به هتل
- هر واحد می‌تواند به صورت جداگانه حل شود. این کار مسائل را ساده‌تر می‌کند.
  - پیدا کردن یک تاکسی تلفنی برای رفتن به فرودگاه، یک آژانس هواپیمایی برای رفتن به اسپانیا و یک اتوبوس شاتل برای رفتن به هتل

واحدبندی

- می‌توان یک سلسله مراتب از واحدها داشت:
  - رفتن به فرودگاه مراحلی دارد:
    - پیدا کردن شماره یک تاکسی تلفنی
    - رزرو تاکسی
    - تنظیم زنگ ساعت برای بیدار شدن
      - تنظیم زنگ ساعت مراحلی دارد...
- نکات کلیدی
  - تقسیم یک مسئله به مسائل کوچکتر مسائل را ساده می‌کند (کنترل پیچیدگی).
  - واحدهای مستقل راحت‌تر حل می‌شوند.

طراحی بالا به پایین

- با این که زنگ ساعت را چگونه تنظیم کنیم شروع نکنید.
- با این که چگونه به فرودگاه برویم شروع نکنید.
- به جای آن از این جزئیات عبور کنید:
  - بعدا در مورد اینکه چگونه زنگ ساعت را تنظیم کنید فکر خواهید کرد.
  - با تصمیمات با مرتبه بالاتری شروع کنید:
    - کجای اسپانیا می‌خواهید بروید؟ مادرید، مالاگا، گرانادا یا...

طراحی بالا به پایین

- سپس به تصمیمات با مرتبه بالاتر بعدی نگاه کنید:
  - چگونه باید به مالاگا بروم؟ پرواز با کدام شرکت هواپیمایی؟ از کدام فرودگاه؟
  - باز از این مورد که چگونه به فرودگاه بروم عبور می‌کنیم.
- حالا رفتن به فرودگاه را برنامه ریزی می‌کنیم.
  - کماکان از اینکه چگونه ساعت را تنظیم کنیم عبور می‌کنیم.
- در نهایت ساعت را تنظیم می‌کنیم.

- کارکردن از بالا به پایین مانند این مثال معمولا یک استراتژی مناسب است.

تجرید

- عبور از جزئیات یعنی: فرض می‌کنیم بعدا آنرا حل خواهیم کرد.
  - تجرید به معنی در نظر نگرفتن بخش‌هایی از مسئله نیست.
- تجرید به معنی در نظر نگرفتن پیاده‌سازی واحدهای دیگر زمانی که در حال پیاده‌سازی یک واحد هستیم است.

تجرید

- در حالی که پیاده‌سازی واحدها را به تعویق می‌اندازیم تصمیم می‌گیریم:
  - مسئله دارای چه واحدهایی است.
  - هر واحد چه کاری انجام می‌دهد. (مثلا ما را به فرودگاه می‌برد)
  - هر واحد به چه چیزی نیاز دارد (مثلا باید قبل از پرواز در فرودگاه باشیم)
  - به این مورد واسط (`interface`) واحد می‌گوییم در مقابل پیاده‌سازی آن.

می توانیم از (واسط یک) واحد در طرح‌ریزی‌مان استفاده کنیم.

می‌دانم که به نحوی به فرودگاه خواهم رفت

تجرید

- اگر بخواهیم کدها را در مرحله اولیه بنویسیم ممکن است چیزی است شبیه به:

```
public void goToSpain(){
  goToAirport();
  flyToSpain();
  goToHotel();
}
```

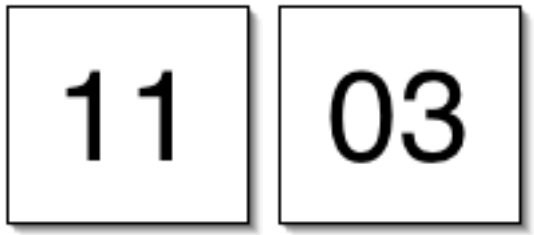
- در این حالت `goToSpain()` را پیش از سایر متدها (واحدها) نوشته‌ایم.
- در این حالت پیاده‌سازی سایر واحدها را به عقب انداخته‌ایم.


Some people feel the rain. Others just get wet.
Bob Marley
quotefancy

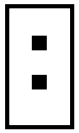
واحدبندی نمایش ساعت

یک نمایش چهار رقمی؟

یا دو نمایش دو رقمی؟



و یک بیت برای اتصال...



- **طراحی:** تصمیم‌گیری در مورد اینکه از چه کلاس‌هایی باید استفاده کنیم و این کلاس‌ها چگونه با هم تعامل دارند – یکی از مشکل‌ترین مراحل در OOP (OOAD)

پیاده‌سازی ساعت دیجیتال

- استفاده از ۲ کلاس:
  - `NumberDisplay`، نمایش اعداد دو رقمی را پیاده‌سازی می‌کند.
  - `ClockDisplay`، ساعت را با استفاده از دو شی `NumberDisplay` پیاده‌سازی می‌کند.
- در اصلاحات شی‌گرا:
  - مسئله را در دو قسمت واحدبندی می‌کنیم.
  - نمایش اعداد را در یک کلاس و نمایش ساعت را در کلاس دیگری محصور می‌کنیم.
  - از `NumberDisplay` برای یک نمایش چهار رقمی استفاده مجدد می‌کنیم.
  - می‌توانیم کدهای `abstract` بنویسیم.
  - از دو کلاس نوشته شده می‌توان در برنامه‌های دیگری هم استفاده کرد.
  - صراحت در شرح طراحی با استفاده از مفاهیم شی‌گرایی به شما (و دیگران) در فهم آنچه دارد می‌گذرد کمک می‌کند.

پیاده‌سازی: NumberDisplay

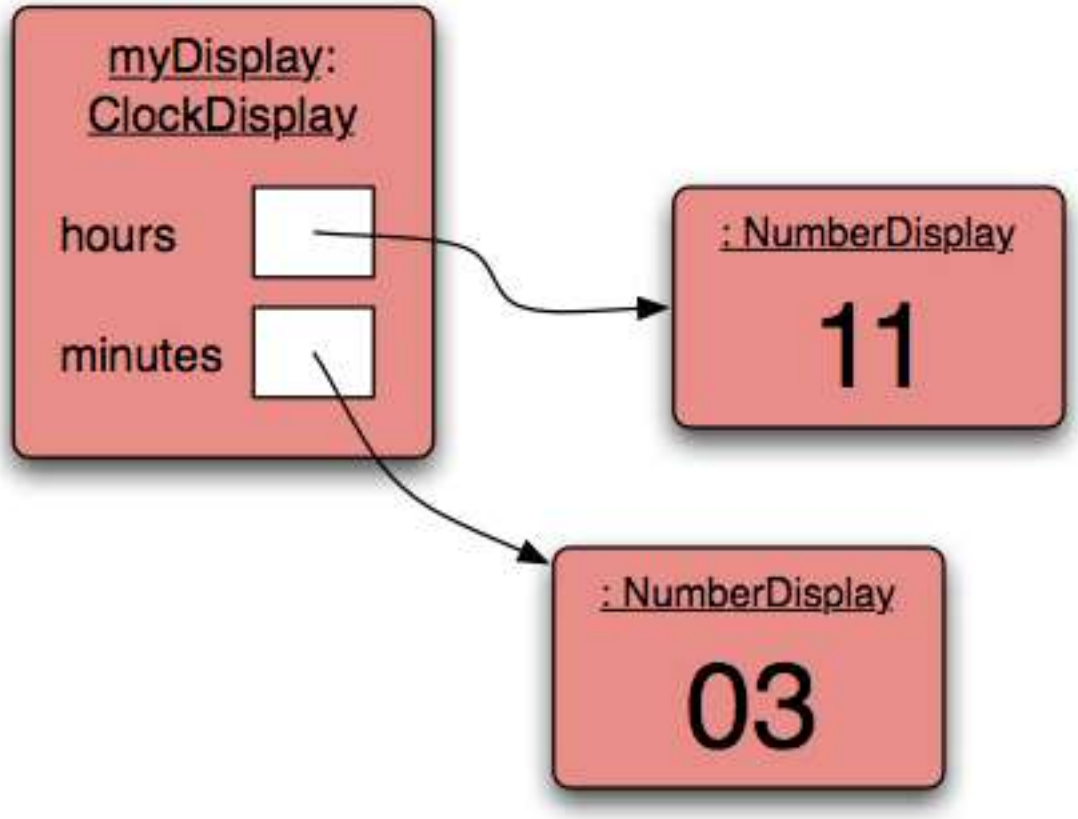
```
public class NumberDisplay
{
    private int limit;
    private int value;

    constructor omitted
    methods omitted
}
```

پیاده‌سازی: ClockDisplay

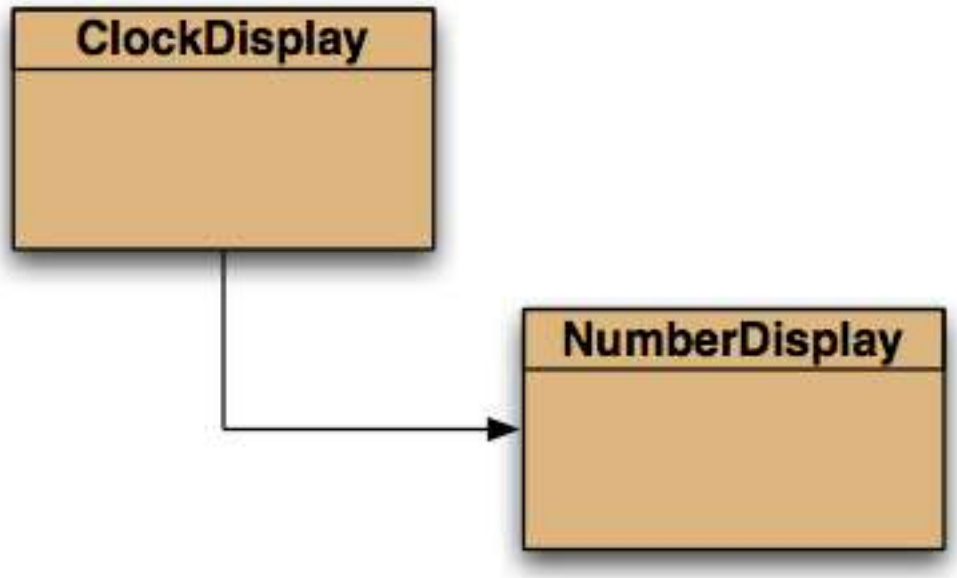
```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;

    constructor omitted
    methods omitted
}
```

- توجه کنید که از کلاس NumberDisplay به عنوان نوع متغیر استفاده شده است.

دیاگرام شی (Object diagram)



دیاگرام کلاس (Class diagram)



- فلش مشخص می‌کند که کلاس `ClockDisplay` از کلاس `NumberDisplay` استفاده می‌کند (`uses`).

دیاگرام کلاس و شی

- دیاگرام کلاس ایستا است: هنگام اجرای برنامه تغییر نمی‌کند.
- دیاگرام شی پویا است: با اضافه یا حذف شدن یک شی هنگام اجرای برنامه تغییر کند.
- در `BlueJ` فقط دیاگرام کلاس نمایش داده می‌شود.
- لازم است نحوه ترسیم و قواعد کلی دیاگرام شی را بدانید.

نوع‌های اولیه/مقداری (Primitive/Value Types)

- برای کارایی بیشتر، جاوا نوع‌های داده‌ای ساده را به عنوان نوع‌های اولیه ذخیره می‌کند.
  - `int, byte, short, long, double, float, char, boolean`
  - نوع‌های اولیه، توکار (`built-in`) هستند. نمی‌توان نوع داده اولیه جدید تعریف کرد.
  - شی نیستند!
    - نمی‌توان به عنوان مثال یک متد به آنها منتسب کرد.

نوع‌های شی/ارجاعی (Object/Reference Types)

- جاوا دارای نوع شی‌های توکار زیادی است (مثلا `String`) و شما می‌توانید نوع‌های جدیدی نیز ایجاد کنید.
  - هر کلاس یک نوع است.
- چند تفاوت بین نوع اولیه و شی
  - نحوه ذخیره شدن
  - اتفاقی که هنگام انتساب آنها می‌افتد

متغیرهای اولیه در برابر متغیرهای شی

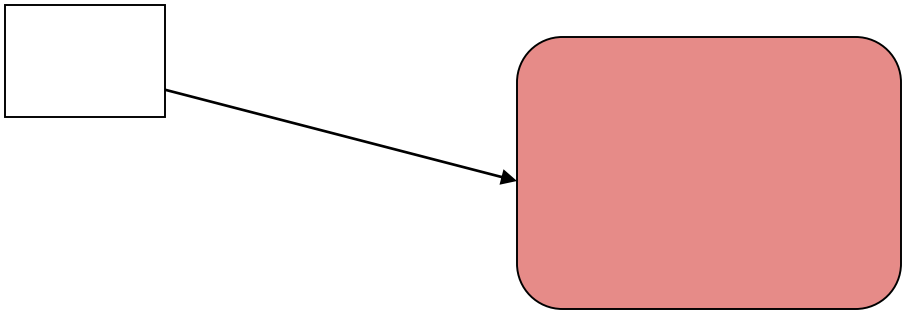
- متغیرهای اولیه مقادیر خود را به صورت داخلی ذخیره می‌کنند.
- متغیرهای شی یک مرجع (reference) به شی را ذخیره می‌کنند.
- در مقایسه با C
  - مرجع همانند اشاره‌گر در C است.

متغیرهای اولیه در برابر متغیرهای شی

```
int i;
```

نوع اولیه

```
SomeObject obj;
```



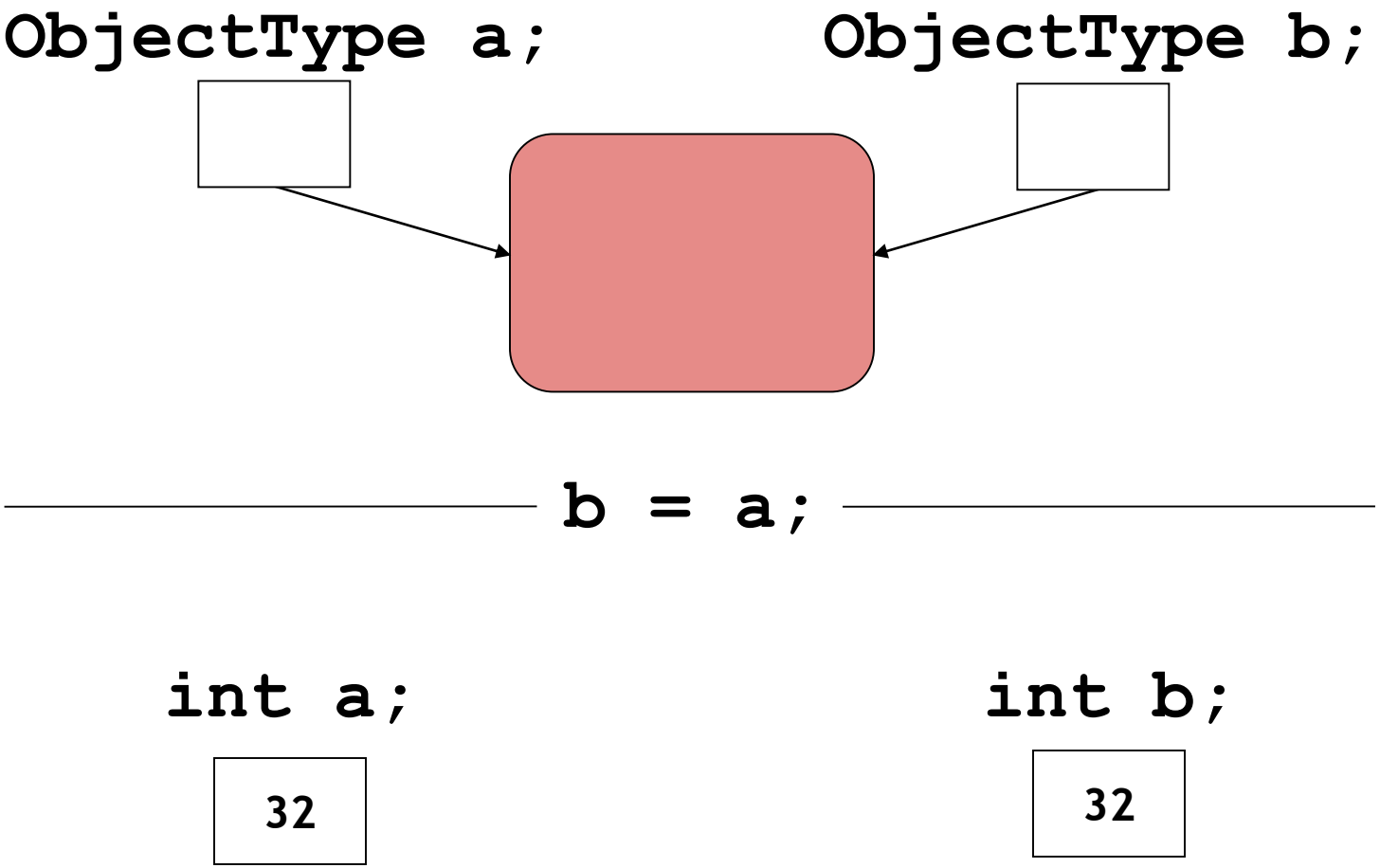
نوع شی

خروجی را مشخص کنید:

```
int a;
int b;
a = 32;
b = a;
a = a + 1;
System.out.println(b);
```

```
Person a;
Person b;
a = new Person("Everett");
b = a;
a.changeName("Delmar");
System.out.println(b.getName());
```

متغیرهای اولیه در برابر متغیرهای شی



متغیرهای اولیه در برابر متغیرهای شی

- نتیجه:
  - هنگامی که متغیر `b` را به `a` منتسب می‌کنید، مقدار متغیر `b` در `a` کپی می‌شود.
  - هنگامی که شی `b` را به `a` منتسب می‌کنید، فقط مرجع کپی می‌شود.

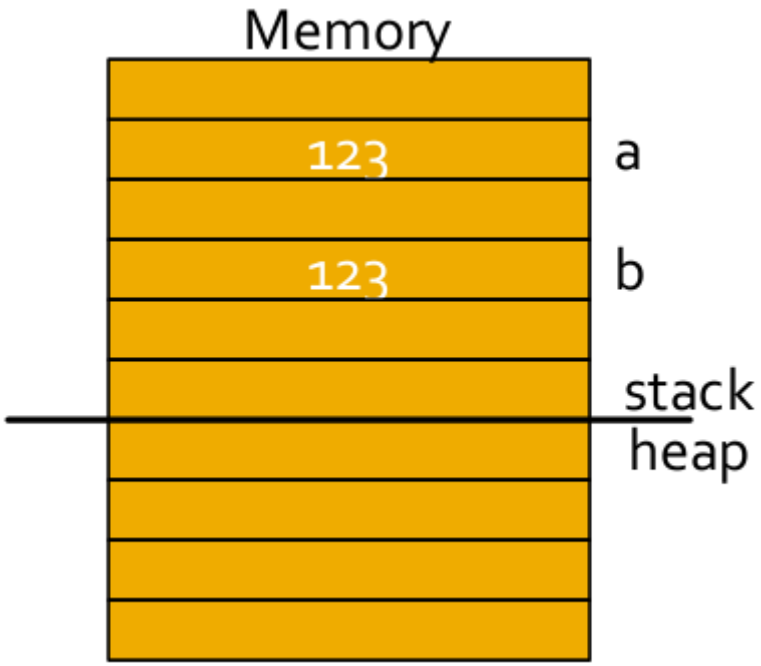
دو مرجع به یک شی خواهیم داشت.

- برای کپی کردن شی، کار دیگری باید انجام داد!

قالب حافظه: نوع‌های مقداری

- مقدار واقعی را در خود ذخیره می‌کند به مکان حافظه را.
- کپی کردن متغیرهای مقداری، یک کپی واقعی ایجاد می‌کند.

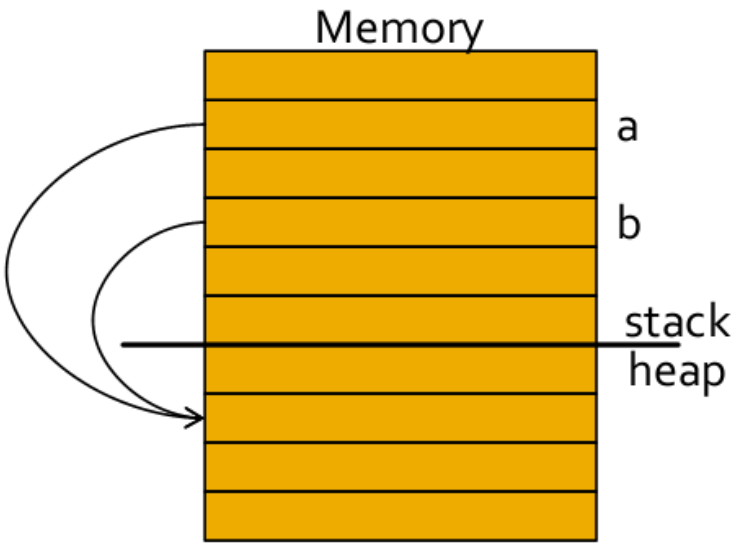
```
int a = 123;
int b = a;
```



قالب حافظه: نوع‌های ارجاعی

- ارجاع به یک مکان از حافظه
  - بسیار شبیه به اشاره‌گرها در زبان‌هایی همانند C/C++.
- می‌تواند با null تنظیم شود:

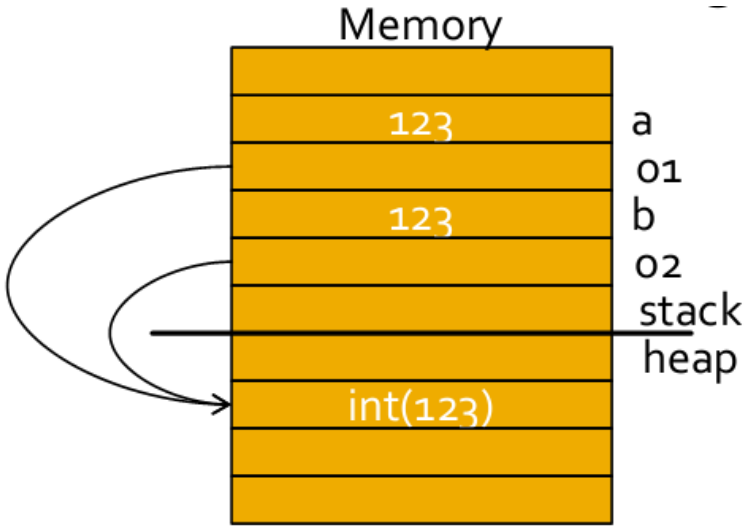
```
TypeA a = new TypeA();
TypeA b = a;
```



Boxing و Unboxing

- نوع‌های مقداری شی نیستند.
  - این امر در اکثر موارد کارآیی را افزایش می‌دهد.
  - اما متغیرهای مقداری می‌توانند هنگام نیاز به شی تبدیل شوند.
  - به این امر Boxing گفته می‌شود. به عمل عکس Unboxing گفته می‌شود.

```
int a = 123;
object o1 = a;
object o2 = o1;
int b = (int) o2;
```



کد کلاس NumberDisplay

```
public class NumberDisplay
{
    private int limit;
    private int value;

    public NumberDisplay(int rollOverLimit)
    {
        limit = rollOverLimit;
        value = 0;
    }

    ...
}
```

کد کلاس NumberDisplay

```
public void setValue(int replacementValue)
{
        if((replacementValue >= 0) && (replacementValue < limit)) {
            value = replacementValue;
        }
}
```

کد کلاس NumberDisplay

```
public void increment()
{
    value = (value + 1) % limit;
}
```

- % اپراتو باقیمانده است: باقیمانده تقسیم اعداد صحیح را بازمی‌گرداند.

- `15 % 12 = 3`
- `15:00 hours is 3pm`

اپراتور باقی‌مانده

- اپراتور (`/`) وقتی روی دو عدد `int` اعمال شود، عمل تقسیم صحیح انجام می‌دهد.
- اپراتور باقیمانده (`%`) باقی‌مانده تقسیم صحیح را باز می‌گرداند.

```
17 / 5 == 3

17 % 5 == 2
```

آزمون

- نتیجه عبارت **(3 % 8)** چیست؟
  - **2**
- تمام مقادیری که ممکن است نتیجه عبارت **(n % 5)** باشد؟
  - **-4, -3, -2, -1, 0, 1, 2, 3, 4**
- تمام مقادیری که ممکن است نتیجه عبارت **(n % m)** باشد؟
  - اعداد در بازه **-(m-1)** و **(m-1)**

کد کلاس NumberDisplay

```
public int getValue(){
    return value;
}

public String getDisplayValue()
{
    if(value < 10) {
        return "0" + value;
    }
    else {
        return "" + value;
    }
}
```

"" یک رشته خالی است. این ترفند موجب تبدیل مقدار بازگشتی به رشته می‌شود تا با نوع بازگشتی مشخص شده هماهنگ باشد.

تا اینجا

- تجرید
- واحدبندی
- تعریف نوع کلاس
- دیاگرام کلاس

- دیاگرام شی
- مرجع شی
- نوع شی
- نوع‌های اولیه

شی ایجاد کننده شی

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        updateDisplay();
    }
}
```

اعلان

تعریف

ایجاد شی

```
private NumberDisplay hours;

hours = new NumberDisplay(24);
```

- از new و به دنبال آن فراخوانی یک سازنده برای ایجاد یک شی جدید استفاده می‌کنیم.

ایجاد شی

```
in class NumberDisplay:
 public NumberDisplay(int rollOverLimit);
```

formal parameter

```
in class ClockDisplay:
 hours = new NumberDisplay(24);
```

actual parameter

ایجاد شی

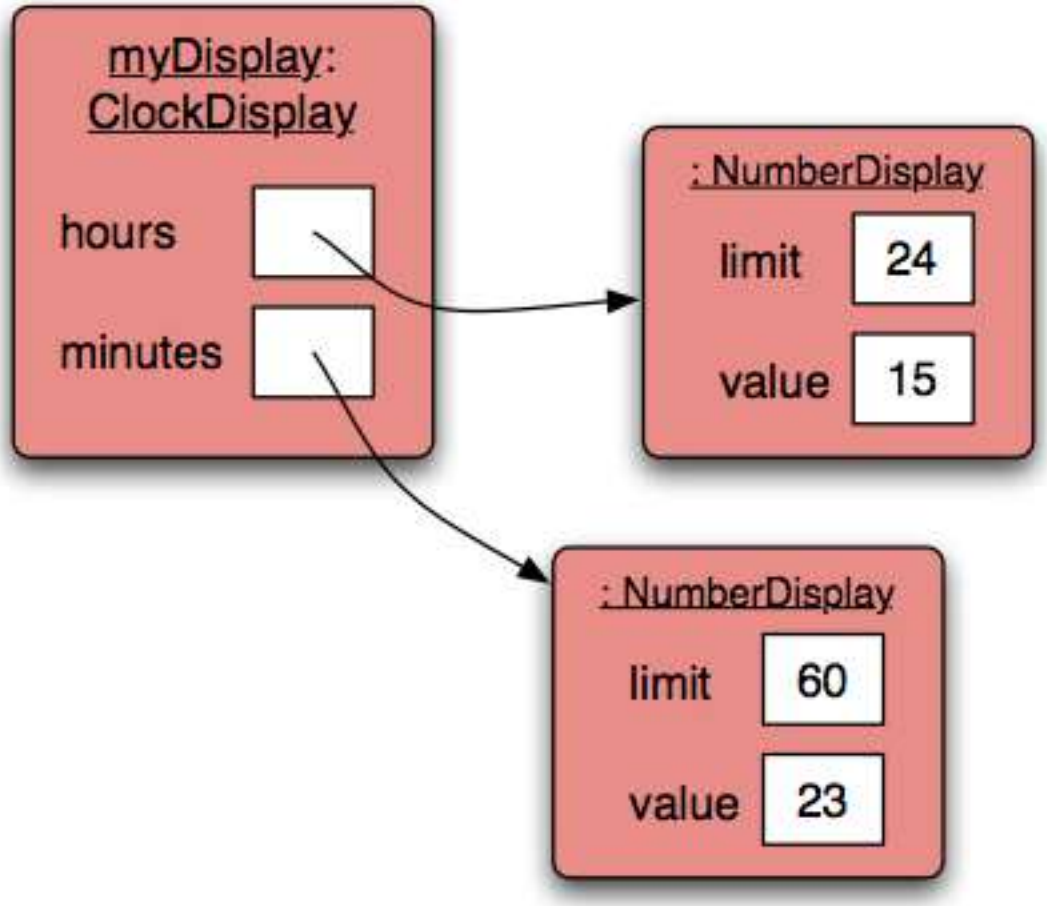
- به سینتکس ویژه مورد استفاده برای ساخت شی از نوع `String` دقت کنید:

```
String myString = “hello”;
```

و نه

```
String myString = new String(“hello”);
```

شی ایجاد کننده شی – نمودار شی



فراخوانی متد

```
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {
        // it just rolled over!
        hours.increment();
    }
    updateDisplay();
}
```

فراخوانی متدی از اشیا دیگر

فراخوانی متدی از اشیا دیگر

فراخوانی متدی از همین شی

فراخوانی خارجی متد (external method call)

- فراخوانی متدی از یک شی دیگر (دسترسی باید وجود داشته باشد).
  - `minutes.increment();`

```
objectName.methodName(parameter-list);
```

فراخوانی داخلی متد (internal method call)

- فراخوانی متدی از همان شی (دسترسی همیشه وجود دارد!)

```
methodName(parameter-list)
```

یا

```
this.methodName(parameter-list)
```

فراخوانی متد

- فراخوانی یک متد از شی هم‌نوع (از یک کلاس)، فراخوانی خارجی محسوب می‌شود.
- فراخوانی داخلی یعنی همین شی
- فراخوانی خارجی یعنی هر شی دیگری بدون در نظر گرفتن نوع آن

قلمرو (scope) و this

- از درس قبل می‌دانیم:
  - یک متغیر فقط در قلمرو خودش قابل استفاده است.
  - قلمرو پارامترها همان متدشان است.
  - قلمرو فیلدها کلاس‌شان است.
- چه اتفاقی خواهد افتاد اگر پارامتر و فیلد همنام باشند؟

```
public class MailItem {
   private String from;
   private String to;

   public MailItem(String from, String to){
        from = ...;
   }
}
```

کدام from؟ پارامتر یا فیلد؟

قلمرو (scope) و this

- جاوا آخرین بلوکی که متغیر در آن تعریف شده را در نظر می‌گیرد.
  - در مثال اسلاید قبل پارامتر form در نظر گرفته می‌شود.
- اما همچنان می‌توان به فیلد from در این متد دسترسی داشت! با کمک **this**.
- this را بخوانید این شی!

```
public class MailItem {
   private String from;
   private String to;

   public MailItem(String from, String to){
        this.from = from;
        this.to = to;
   }
}
```

ClockDisplay ادامه

```
/**
 * Update the internal string that
 * represents the display.
*/
private void updateDisplay()
{
    displayString =
        hours.getDisplayValue() + ":" +
        minutes.getDisplayValue();
}
```

null

- null یک مقدار ویژه در جاوا است.
- تمام اشیا هنگام اعلان و پیش از تعریف برابر null هستند.

```
private NumberDisplay hours
```

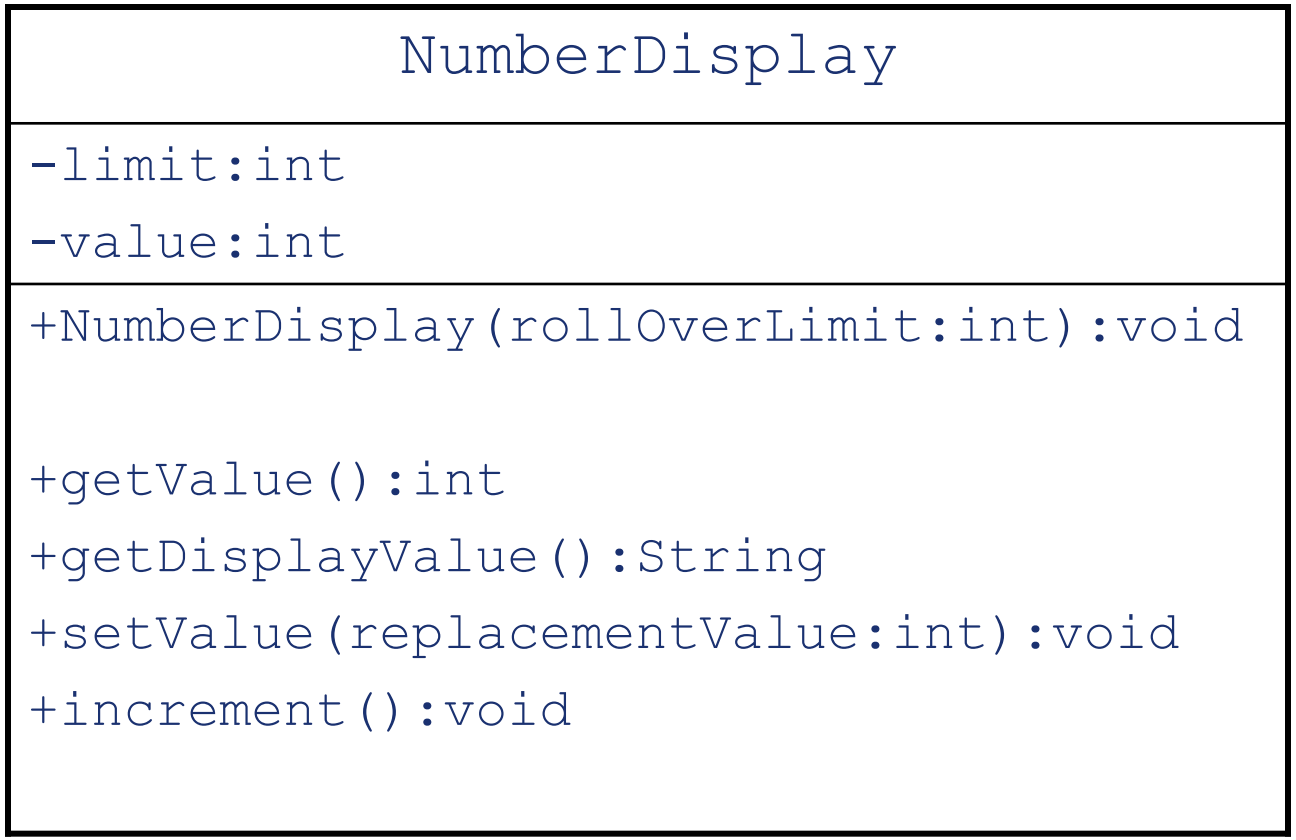
- می‌توان null بودن یک متغیر را مورد بررسی قرار داد.

```
if(hours == null) { ... }
```

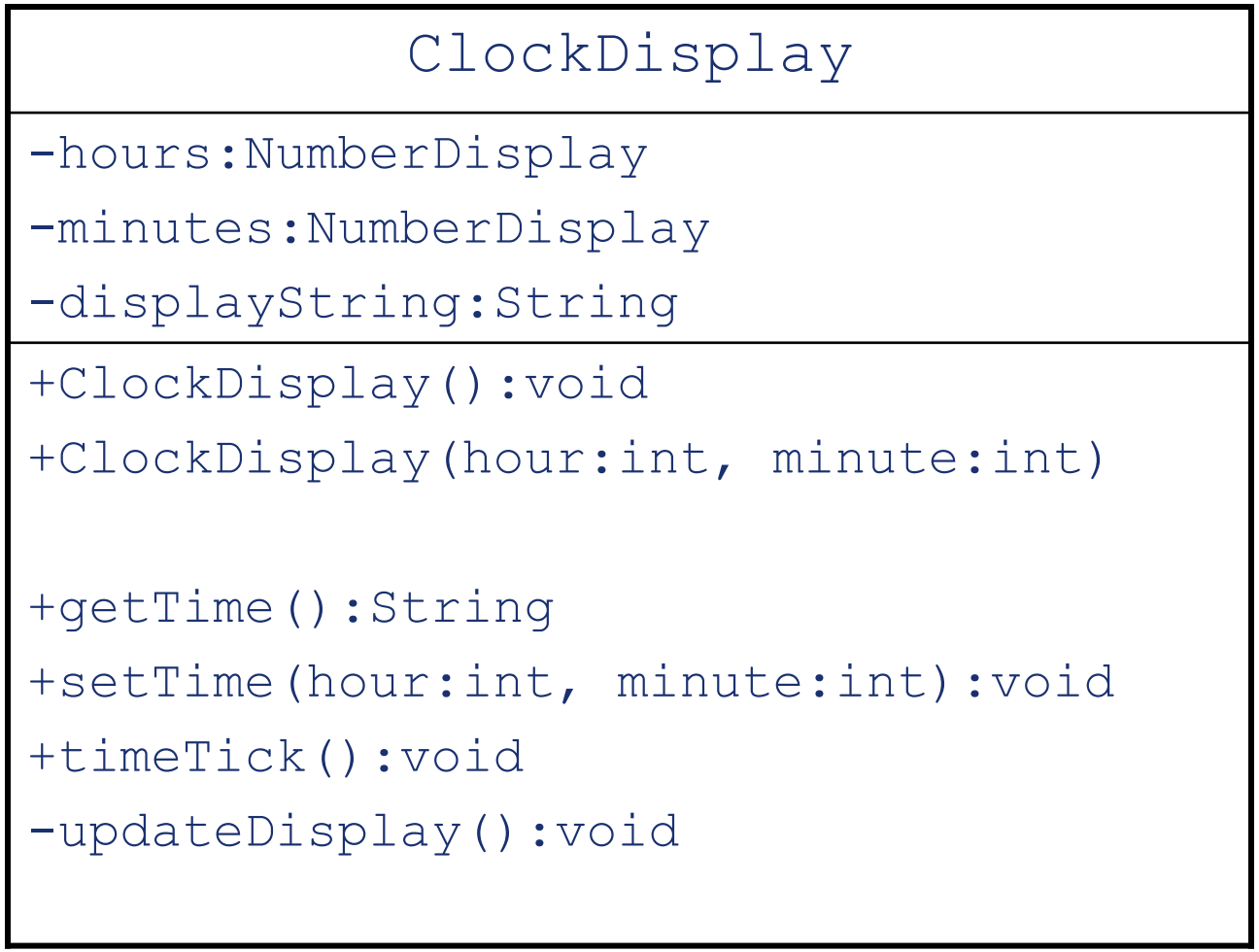
- می‌توان مقدار متغیرها را به null تغییر داد.

```
hours = null;
```

NumberDisplay UML



ClockDisplay UML

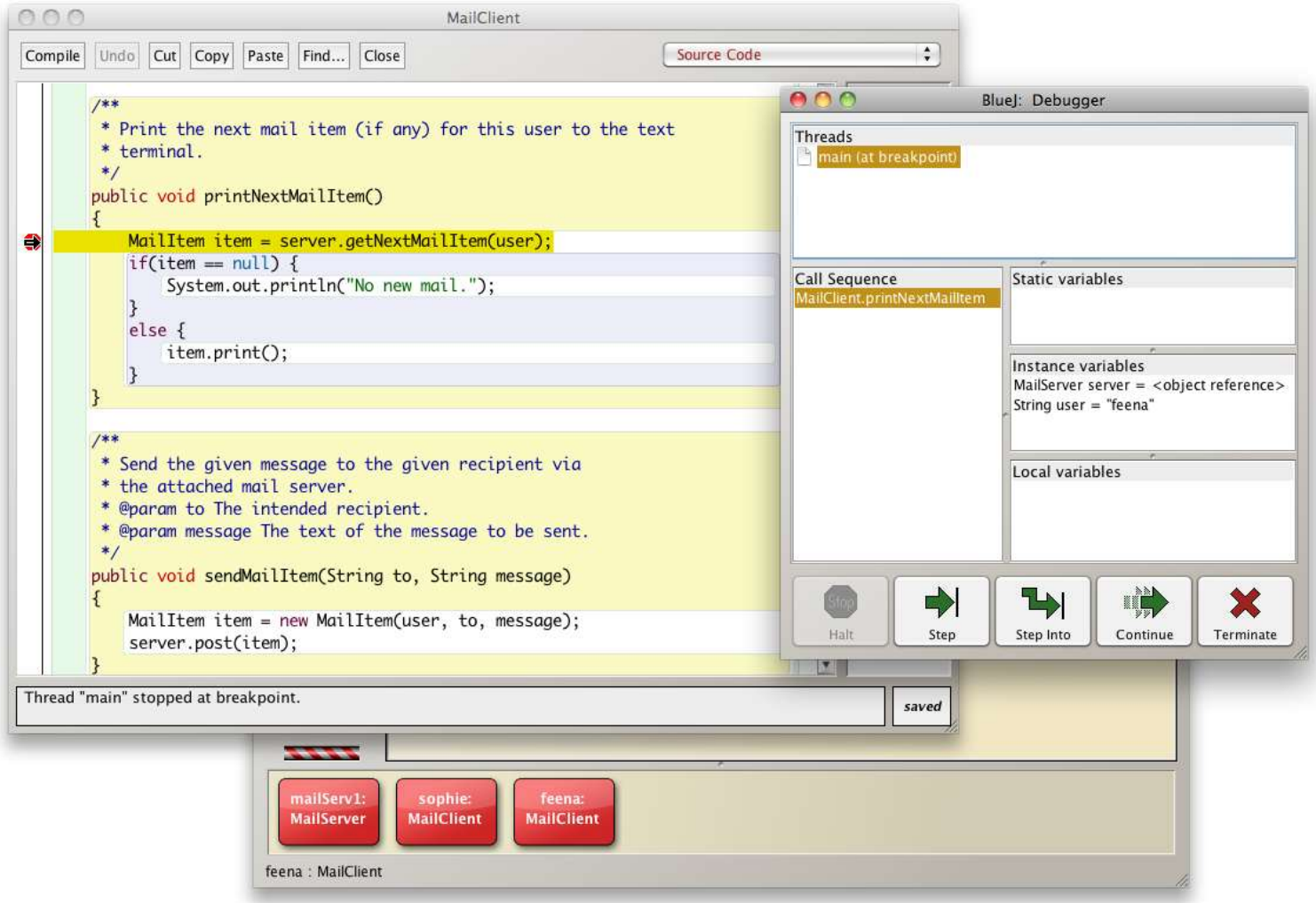


Demo

The debugger

- Useful for gaining insights into program behavior …
- … whether or not there is a program error.
- Set breakpoints.
- Examine variables.
- Step through code.

The debugger



این جلسه

مفاهیم

- abstraction
- modularization
- divide and conquer
- class & object diagrams
- classes define types
- object reference
- object creation
- external method call

جاوا

- Object construction (new)
- method calls (dot notation)
- this
- null