Introduction to Software Testing

(2nd edition)

Chapter 4

Putting Testing First

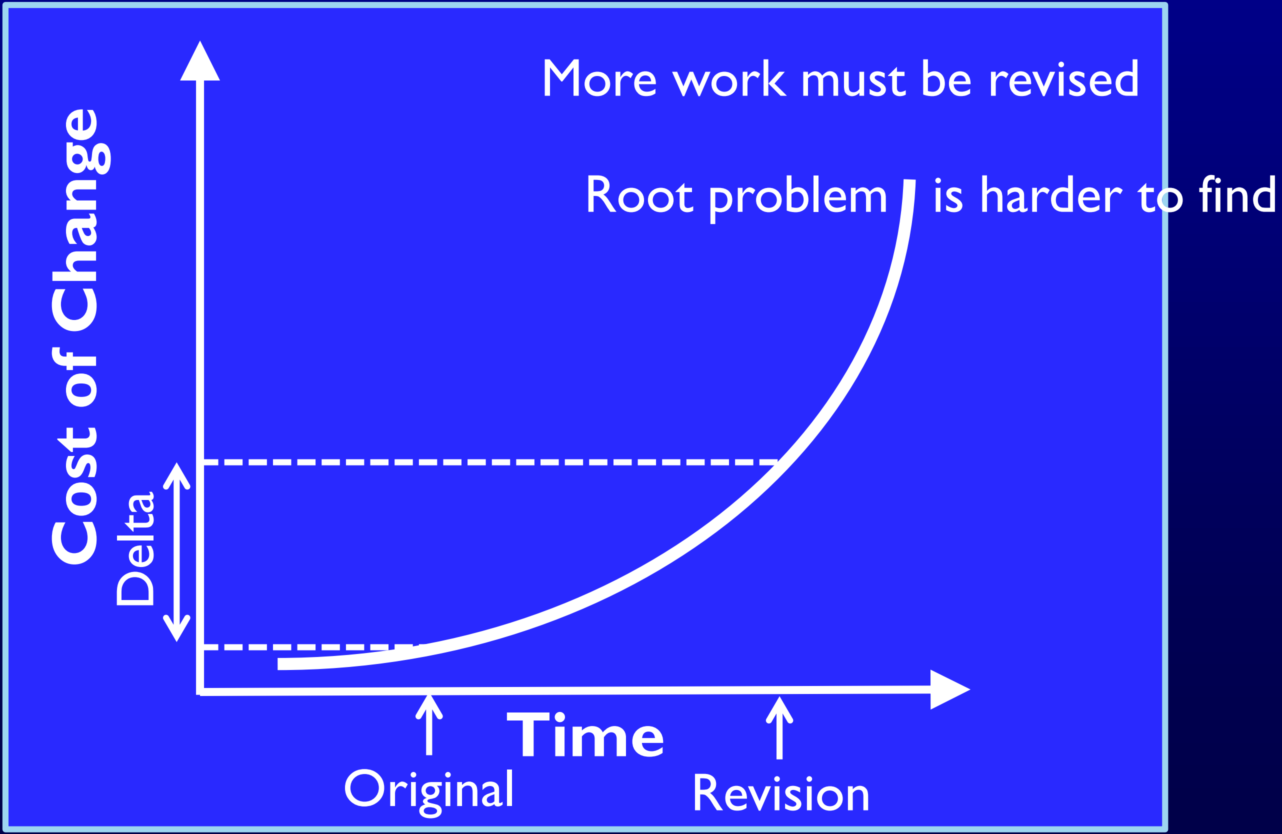
Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

August 2014

The Increased Emphasis on Testing

- Philosophy of traditional software development methods
  - Upfront analysis
  - Extensive modeling
  - Reveal problems as early as possible



Traditional Assumptions

1. Modeling and analysis can identify potential problems early in development

2. Savings implied by the cost-of-change curve justify the cost of modeling and analysis over the life of the project

- These are true if requirements are always complete and current
- But those annoying customers keep changing their minds!
  - Humans are naturally good at approximating
  - But pretty bad at perfecting
- These two assumptions have made software engineering frustrating and difficult for decades

Thus, agile methods …

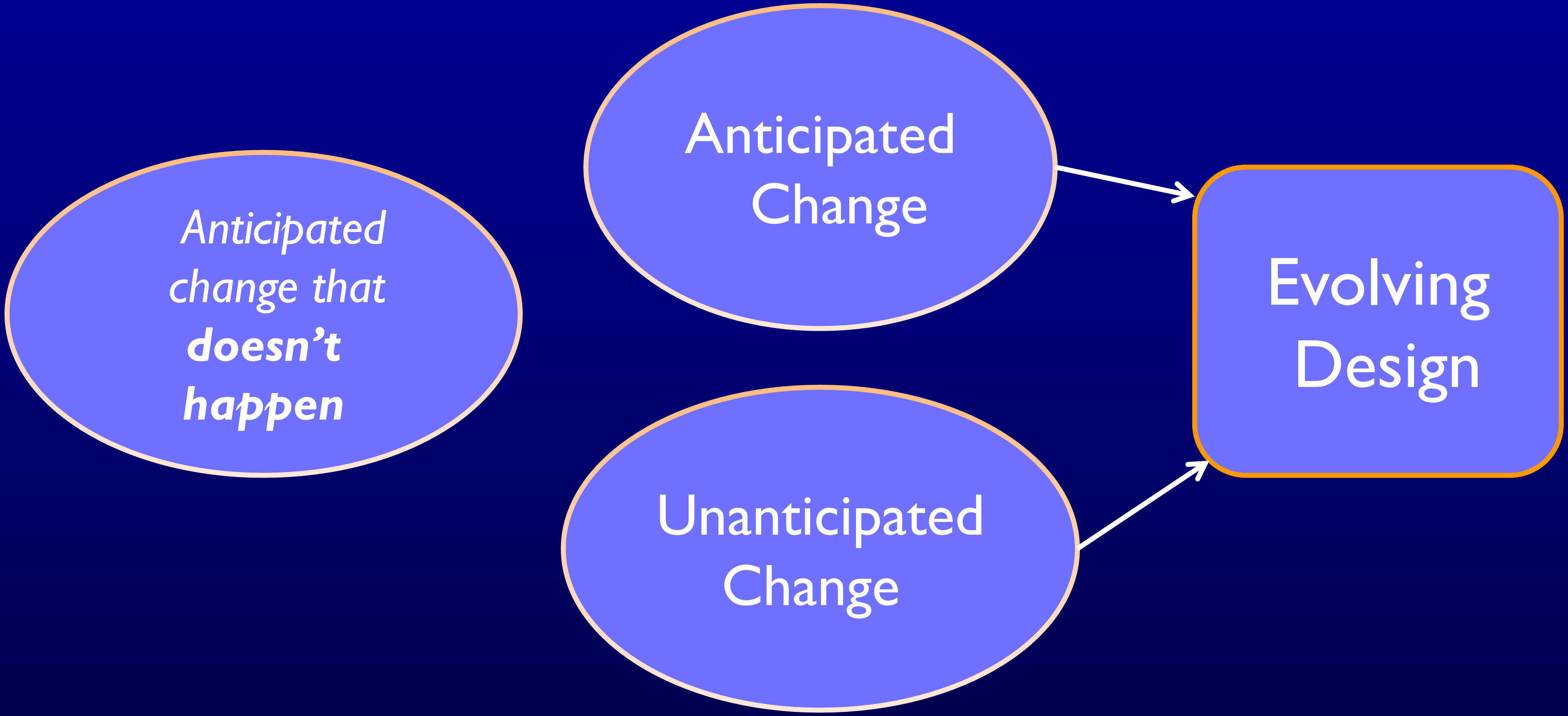
Why Be Agile ?

- Agile methods start by recognizing that neither assumption is valid for many current software projects
  - Software engineers are not good at developing requirements
  - We do not anticipate many changes
  - Many of the changes we do anticipate are not needed
- Requirements (and other "non-executable artifacts") tend to go out of date very quickly
  - We seldom take time to update them
  - Many current software projects change continuously
- Agile methods expect software to start small and evolve over time
  - Embraces software evolution instead of fighting it

Supporting Evolutionary Design

Traditional design advice says to anticipate changes

Designers often anticipate changes that don’t happen



Both anticipated and unanticipated changes affect design

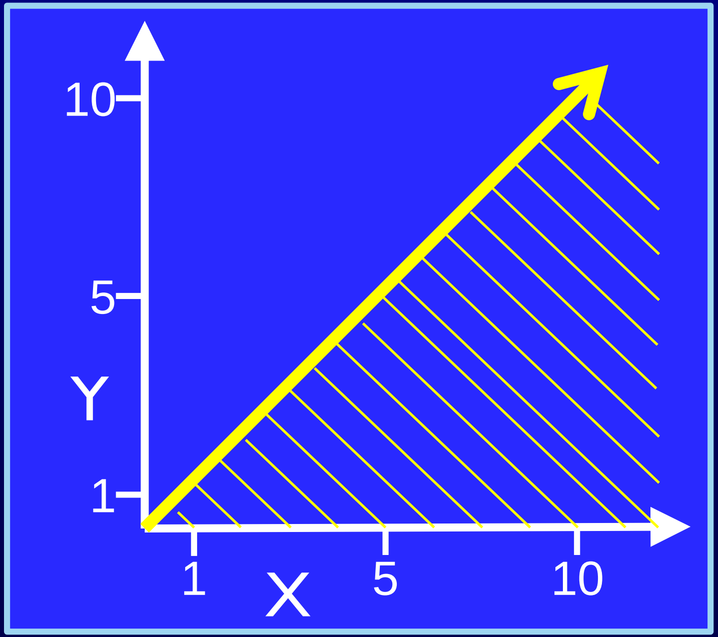
The Test Harness

- An agile principle states that traditional planning is not precise because:
  - Predicting system evolution is fundamentally hard
  - Hence expected savings from the cost-of-change curve do not materialize
- Instead, agile methods such as TDD defer many design and analysis decisions and focus instead on creating a running system that does "something" as early as possible
- At first glance, this may sound like a return to the dark days before traditional software engineering
- But no! In fact, there is a crucial difference
- So, what's different?
- Answer: The test harness

The Test Harness as Guardian (4.2)

What is Correctness ?

Traditional Correctness
(Universal)
$\forall x,y, x \geq y$



Agile Correctness
(Existential)

{ (1, 1) → T
(1, 0) → T
(0, 1) → F
(10, 5) → T
(10, 12) → F }

A Limited View of Correctness

- In traditional methods, we try to define all correct behavior completely, at the beginning
  - What is correctness?
  - Does "correctness" mean anything in large engineering products?
  - People are VERY BAD at completely defining correctness
- In agile methods, we redefine correctness to be relative to a specific set of tests
  - If the software behaves correctly on the tests, it is "correct"
  - Instead of defining all behaviors, we demonstrate some behaviors
  - Mathematicians may be disappointed at the lack of completeness

But software engineers ain't mathematicians!

Test Harnesses Verify Correctness

A *test harness* runs all automated tests efficiently and reports results to the developers

- Tests must be automated
  - Test automation is a prerequisite to test driven development
- Every test must include a test oracle that can evaluate whether that test executed correctly
- The tests replace the requirements
- Tests must be high quality and must run quickly
- We run tests every time we make a change to the software

The Development Cycle in Agile Methods

- In agile methods, test cases are the de facto specification for the system
- This makes testing the central activity in development
- This is the reason that agile methods such as TDD order
  - writing tests first
  - implementing functionality second
  - and following good design principles third (i.e. refactoring)
- It is important to emphasize that good design still matters in TDD
  - It simply occupies a different, and later, niche in the development cycle

Executable vs Non-Executable Documents

- Agile methods attempt to make executable artifacts to satisfy needs
  - Traditionally were satisfied by non-executable artifacts
- The definition of success in agile methods differs from traditional development
- Traditional development defines success as "on time and on budget"
- Agile methods aim first for having something executable available from the very beginning of development
  - Then producing a different, and better, product than the one originally envisioned
- To make agile work, test cases need to be of high quality and test processes need to be efficient

Continuous Integration

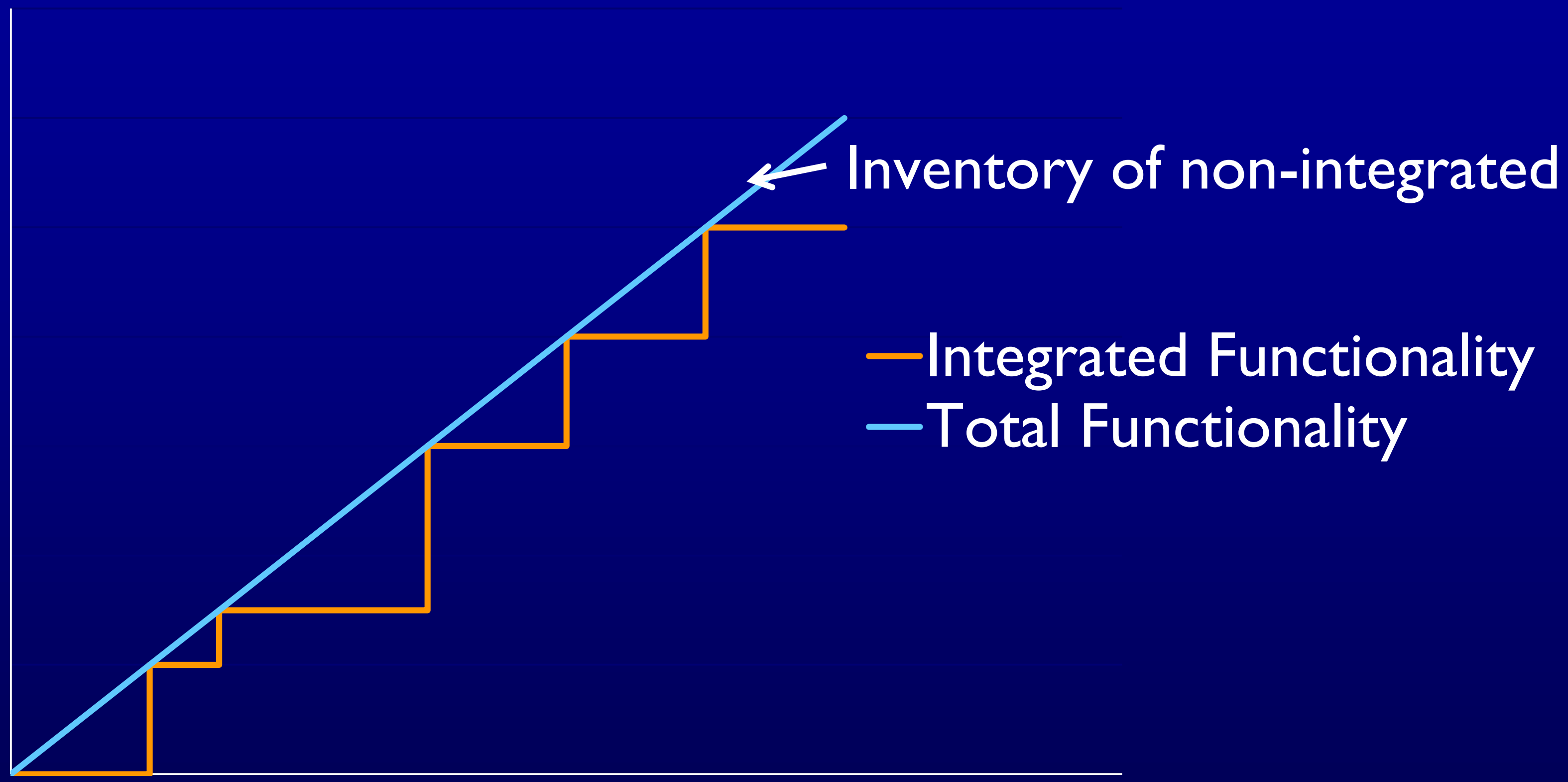
- Agile methods work best when the current version of the software can be run against all tests at any time

> A *continuous integration server* rebuilds the system and reverifies tests whenever *any* update is checked into the repository

- Mistakes are caught earlier
- Other developers are aware of changes early
- The rebuild and reverify must happen as soon as possible
  - Thus, tests need to execute quickly

> A *continuous integration server* doesn't just run tests, it decides if a modified system is still correct

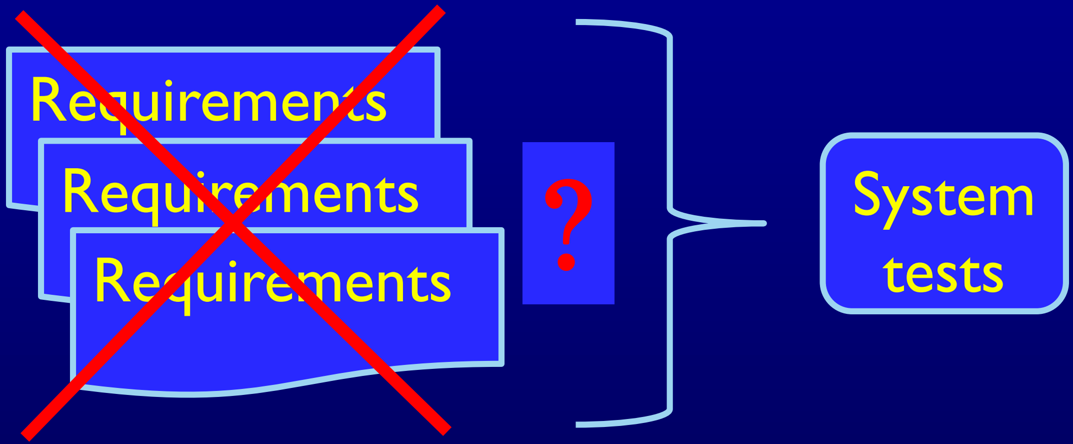
Continuous Integration Reduces Risk



Non-integrated functionality is dangerous!

System Tests in Agile Methods

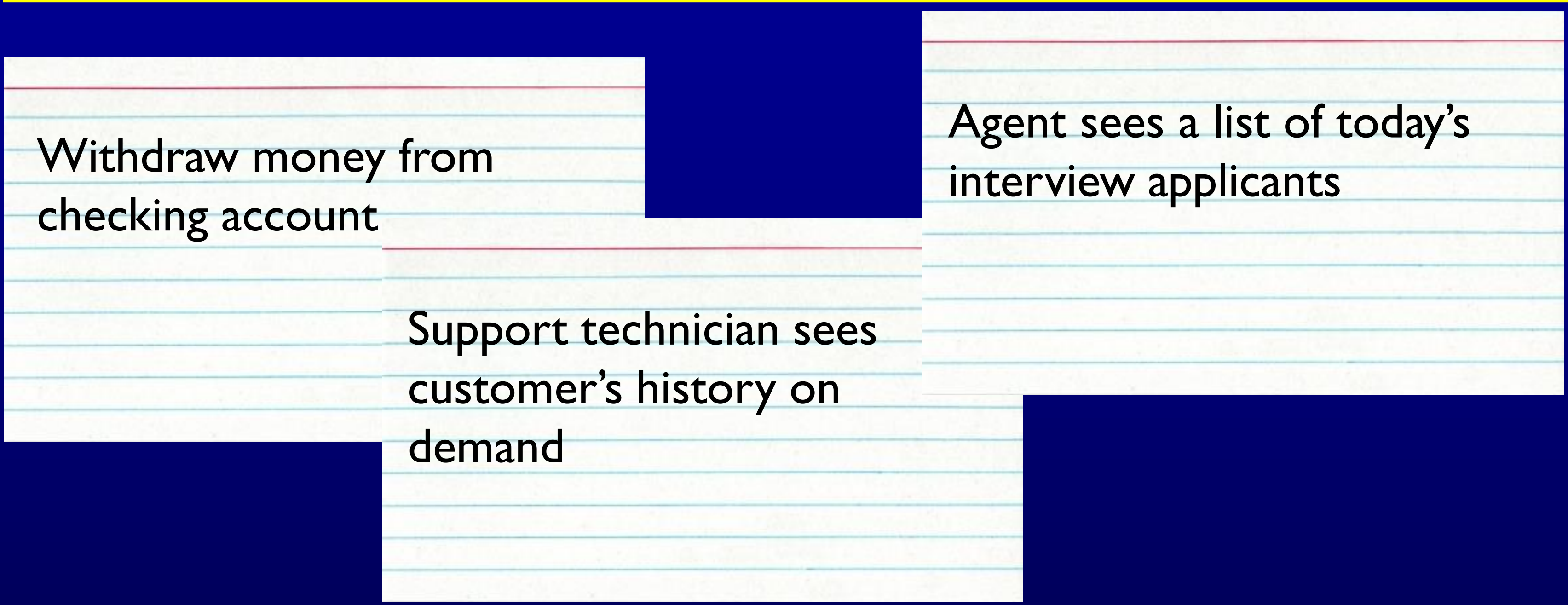
Traditional testers often design system tests from requirements



But … what if there are no traditional requirements documents ?

User Stories

A *user story* is a few sentences that captures what a user will do with the software



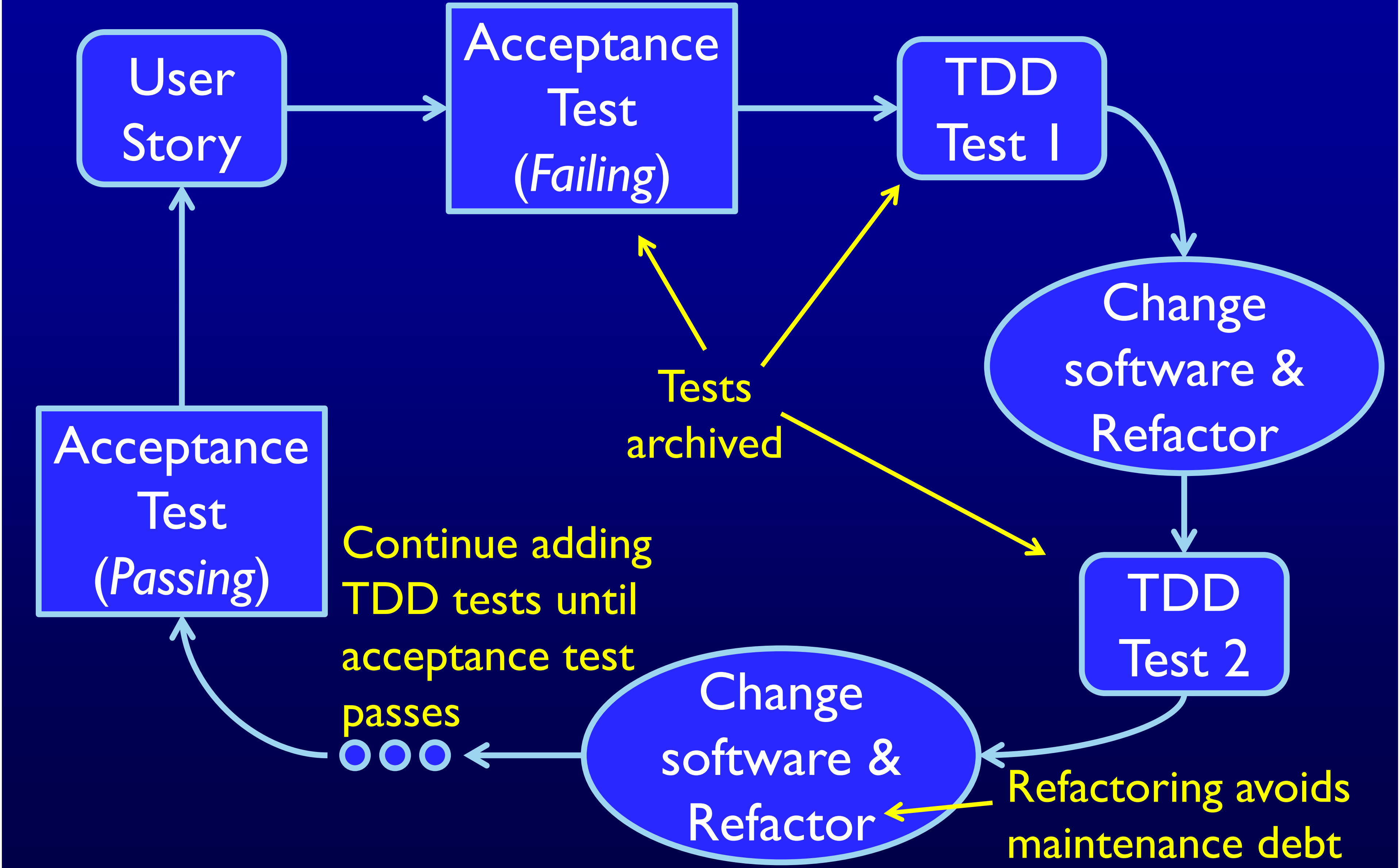
Withdraw money from checking account

Support technician sees customer's history on demand

Agent sees a list of today's interview applicants

- In the language of the **end user**
- Usually small in scale with **few details**
- **Not** archived

Acceptance Tests in Agile Methods



User Story
Acceptance Test (*Failing*)
TDD Test 1
Change software & Refactor
TDD Test 2
Change software & Refactor
Acceptance Test (*Passing*)
Tests archived

Continue adding TDD tests until acceptance test passes

Refactoring avoids maintenance debt

Adding Tests to Existing Systems

- Most of today's software is legacy
  - No legacy tests
  - Legacy requirements hopelessly outdated
  - Designs, if they were ever written down, lost
- Companies sometimes choose not to change software because of fear of the consequences of changes

How to apply TDD to legacy software with no tests?

- Create an entire new test set? — too expensive!
- Give up? — a mixed project is unmanageable

Incremental TDD

- When a change is made, add TDD tests for just that change
  - Refactor
- As the project proceeds, the collection of TDD tests continues to grow
- Eventually the software will have strong TDD tests

The Testing Shortfall

- Do TDD tests (acceptance or otherwise) test the software well?
  - Do the tests achieve good coverage on the code?
  - Do the tests find most of the faults?
  - If the software passes, should management feel confident the software is reliable?

NO!

Why Not?

- Most agile tests focus on "*happy paths*"
  - What should happen under normal use
- They often miss things like
  - Confused-user paths
  - Creative-user paths
  - Malicious-user paths

The agile methods literature does not give much guidance

What Should Testers Do?

Ummm … Excuse me, Professor …

What do I **DO**?

Design Good Tests

1. Use a human-based approach

- Create additional user stories that describe non-happy paths
- How do you know when you're finished?
- Some people are very good at this, some are bad, and it's hard to teach

Part 2 of book …

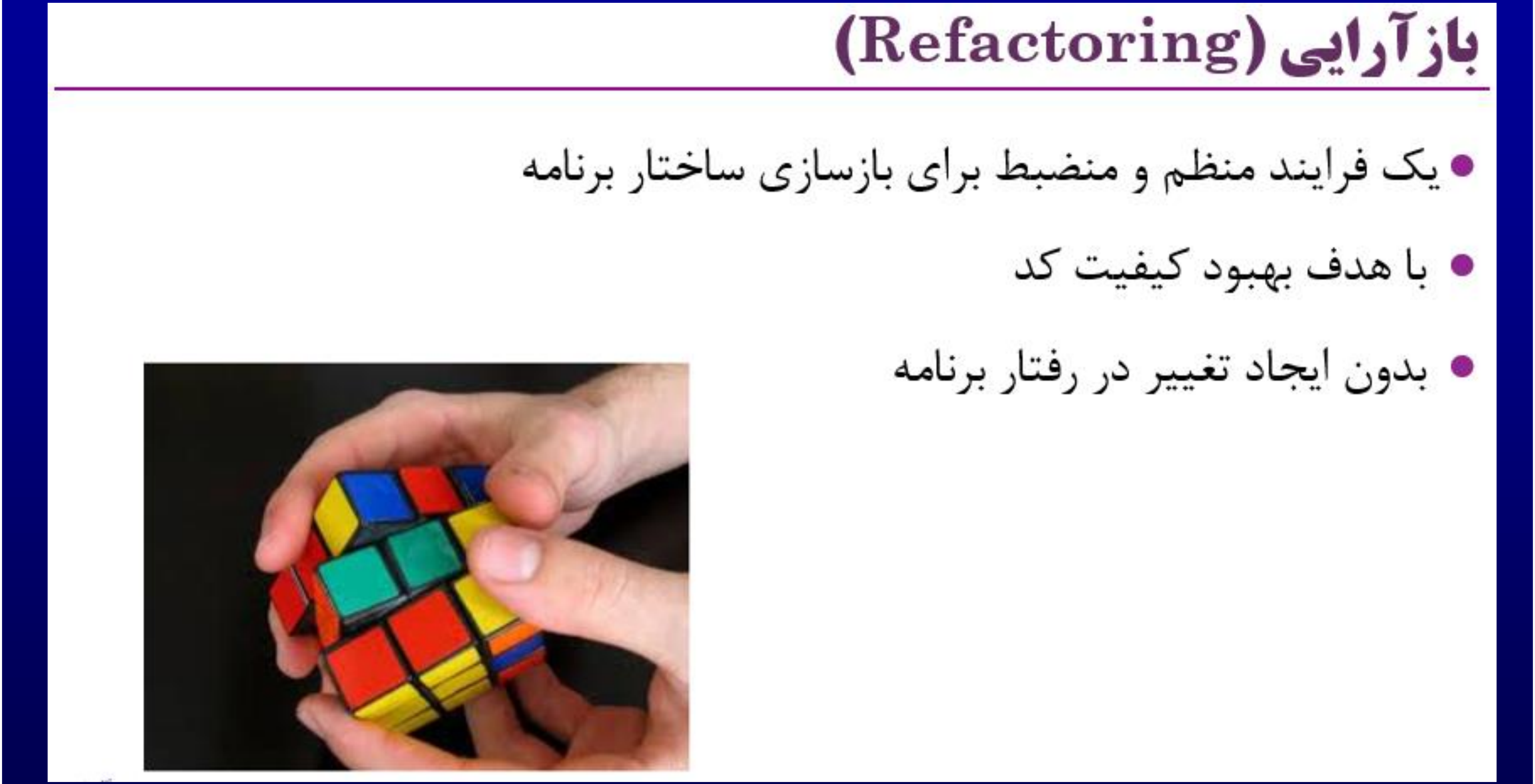
2. Use modeling and criteria

- Model the input domain to design tests
- Model software behavior with graphs, logic, or grammars
- A built-in sense of completion
- Much easier to teach—engineering
- Requires discrete math knowledge

Summary

- More companies are putting testing first
- This can dramatically decrease cost and increase quality
- A different view of "*correctness*"
  - Restricted but practical
- Embraces evolutionary design
- TDD is definitely not test automation
  - Test automation is a prerequisite to TDD
- Agile tests aren't enough
  - Applying coverage criteria can help testers design very high quality tests

مطالب تکمیلی: بازآرایی



بازآرایی (Refactoring)

- یک فرایند منظم و منضبط برای بازسازی ساختار برنامه
- با هدف بهبود کیفیت کد
- بدون ایجاد تغییر در رفتار برنامه

تعریف بازآرایی

- تغییری در ساختار داخلی نرم‌افزار،
- که باعث می‌شود راحت‌تر خوانده و فهمیده شود،
- و تغییر (نگهداری) آن کم هزینه‌تر و ساده‌تر شود،
- بدون این که تغییری در رفتار نرم‌افزار مشاهده شود.
- مهمترین فایده بازآرایی: افزایش **قابلیت نگهداری** نرم‌افزار

بازآرایی چه می‌کند؟

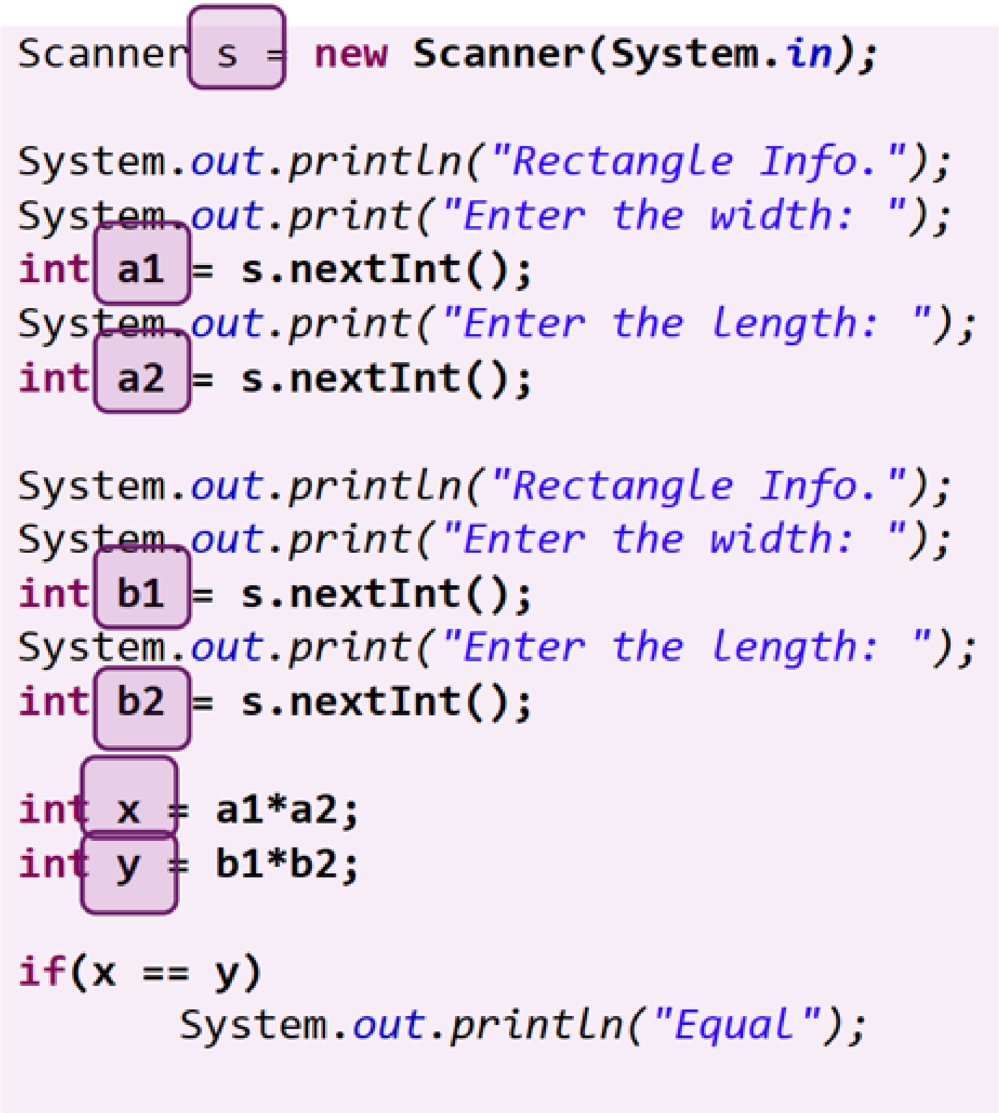
- بهبود ساختار داخلی برنامه
- اجرای فرایندی منظم برای تمیز کردن کد
- بهبود طراحی برنامه بعد از نوشتن کد
  - بخصوص در فرایندهای چابک تولید نرم‌افزار
  - بهبود دائمی طراحی برنامه

فرایند بازآرایی

- در هر مرحله، یک اشکال ساختاری در متن برنامه پیدا می‌کنیم
  - مثلاً یک متد که زیادی طولانی شده است
  - منظور از اشکال، باگ نیست
  - هر یک از این علائم و اشکالات ساختاری، یک «بوی بد» در برنامه هستند
  - Bad Smells
- هر «بوی بد»، با یک تکنیک مشخص برطرف می‌شود
  - تکنیک‌های بازآرایی (Refactoring Techniques)

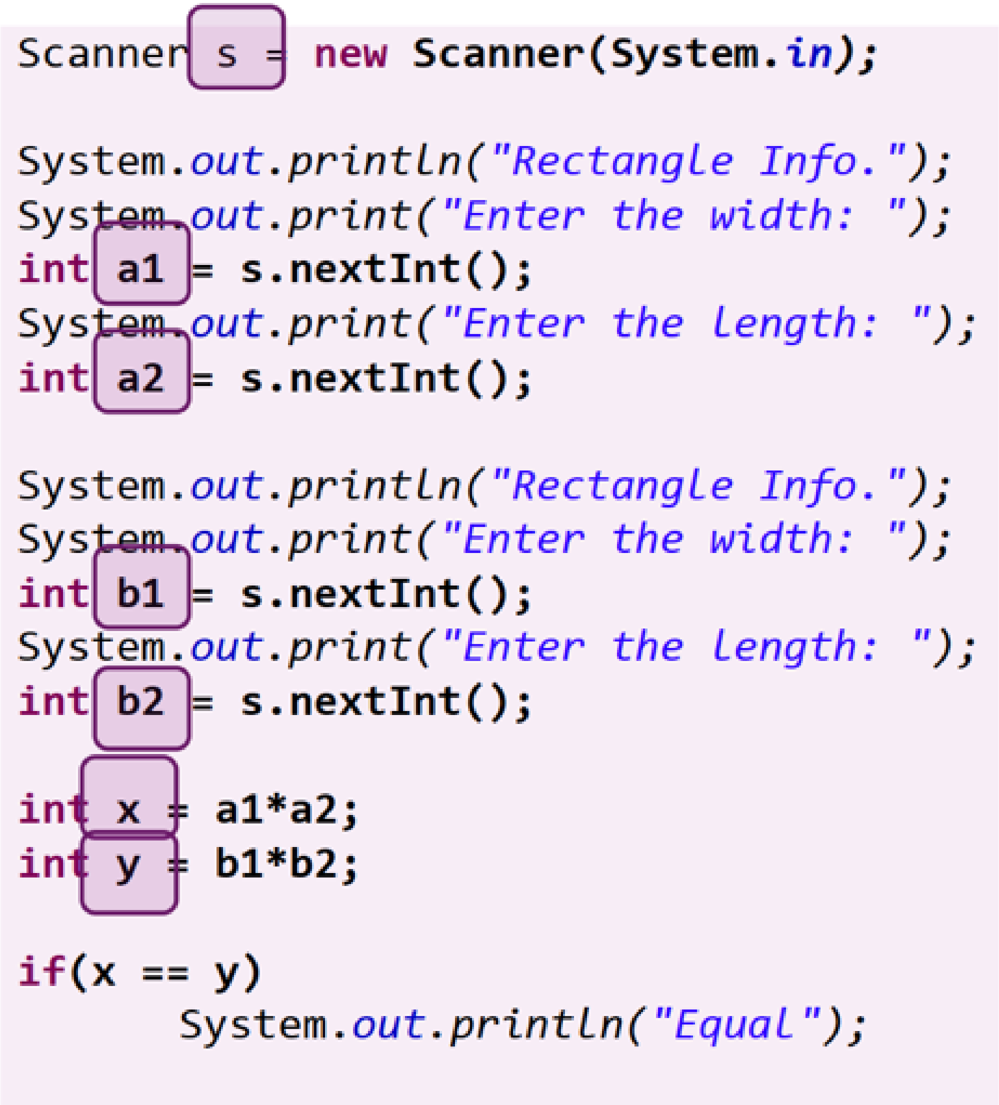
مثال

- این برنامه را ببینید
- چه اشکالاتی دارد؟
- چگونه ساختار آن را بهبود بخشیم؟



```java
Scanner s = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int a1 = s.nextInt();
System.out.print("Enter the length: ");
int a2 = s.nextInt();

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int b1 = s.nextInt();
System.out.print("Enter the length: ");
int b2 = s.nextInt();

int x = a1*a2;
int y = b1*b2;

if(x == y)
        System.out.println("Equal");
```

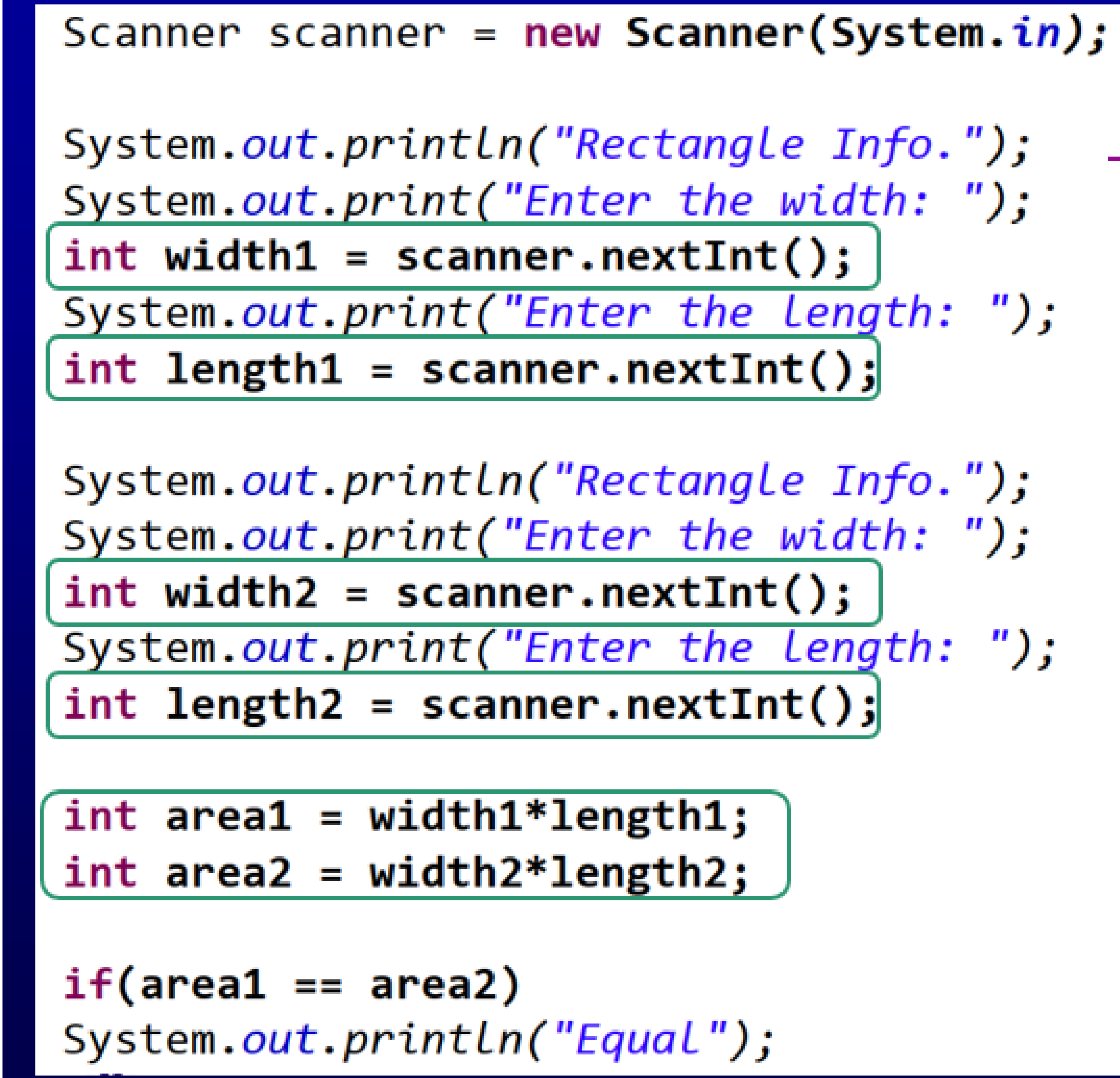
مثال



```
Scanner s = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int a1 = s.nextInt();
System.out.print("Enter the length: ");
int a2 = s.nextInt();

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int b1 = s.nextInt();
System.out.print("Enter the length: ");
int b2 = s.nextInt();

int x = a1*a2;
int y = b1*b2;

if(x == y)
        System.out.println("Equal");
```

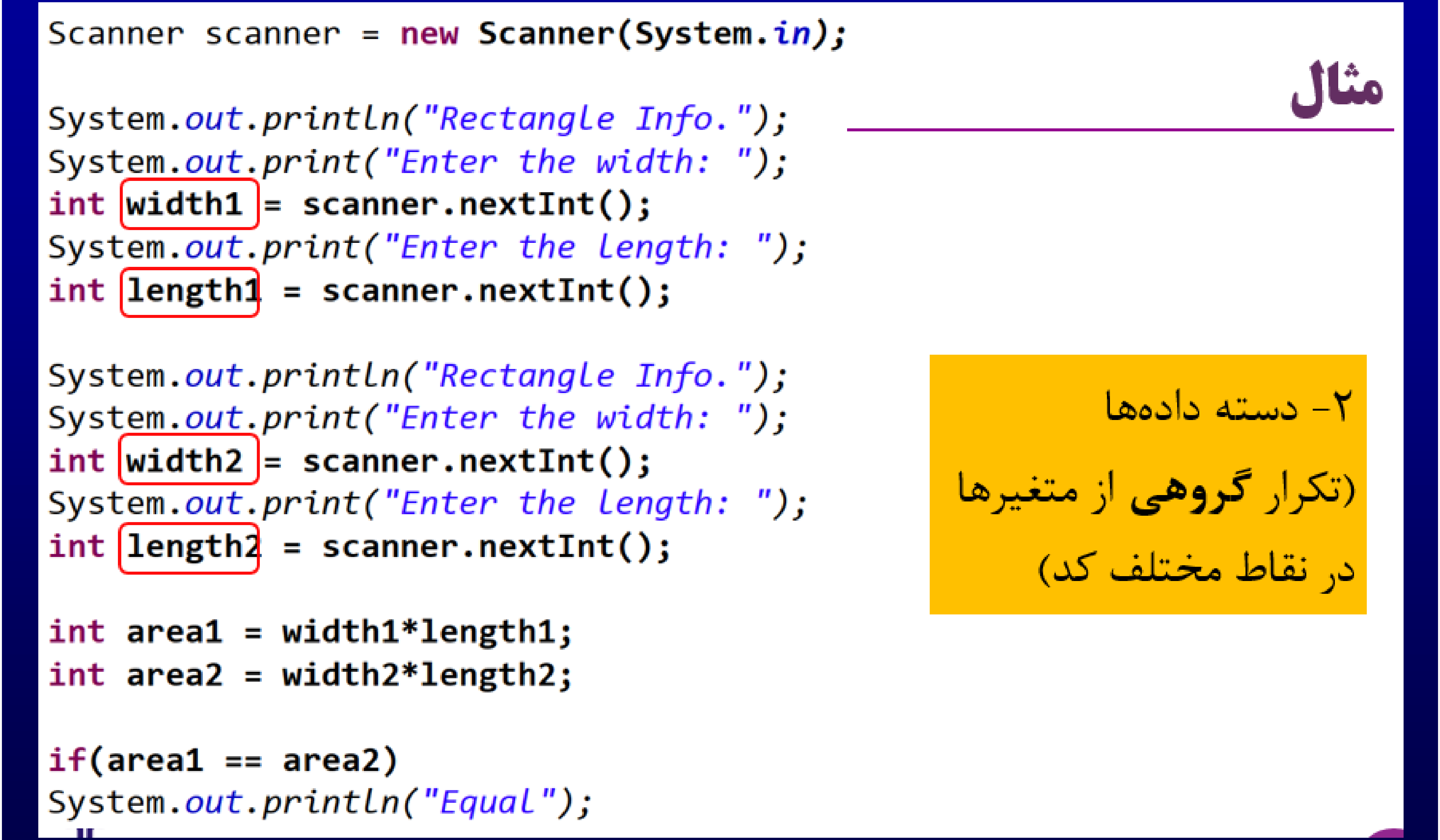
۱ – اسامی نامناسب برای متغیرها

مثال



```
Scanner scanner = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width1 = scanner.nextInt();
System.out.print("Enter the length: ");
int length1 = scanner.nextInt();

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width2 = scanner.nextInt();
System.out.print("Enter the length: ");
int length2 = scanner.nextInt();

int area1 = width1*length1;
int area2 = width2*length2;

if(area1 == area2)
System.out.println("Equal");
```

تکنیک تغییر نام



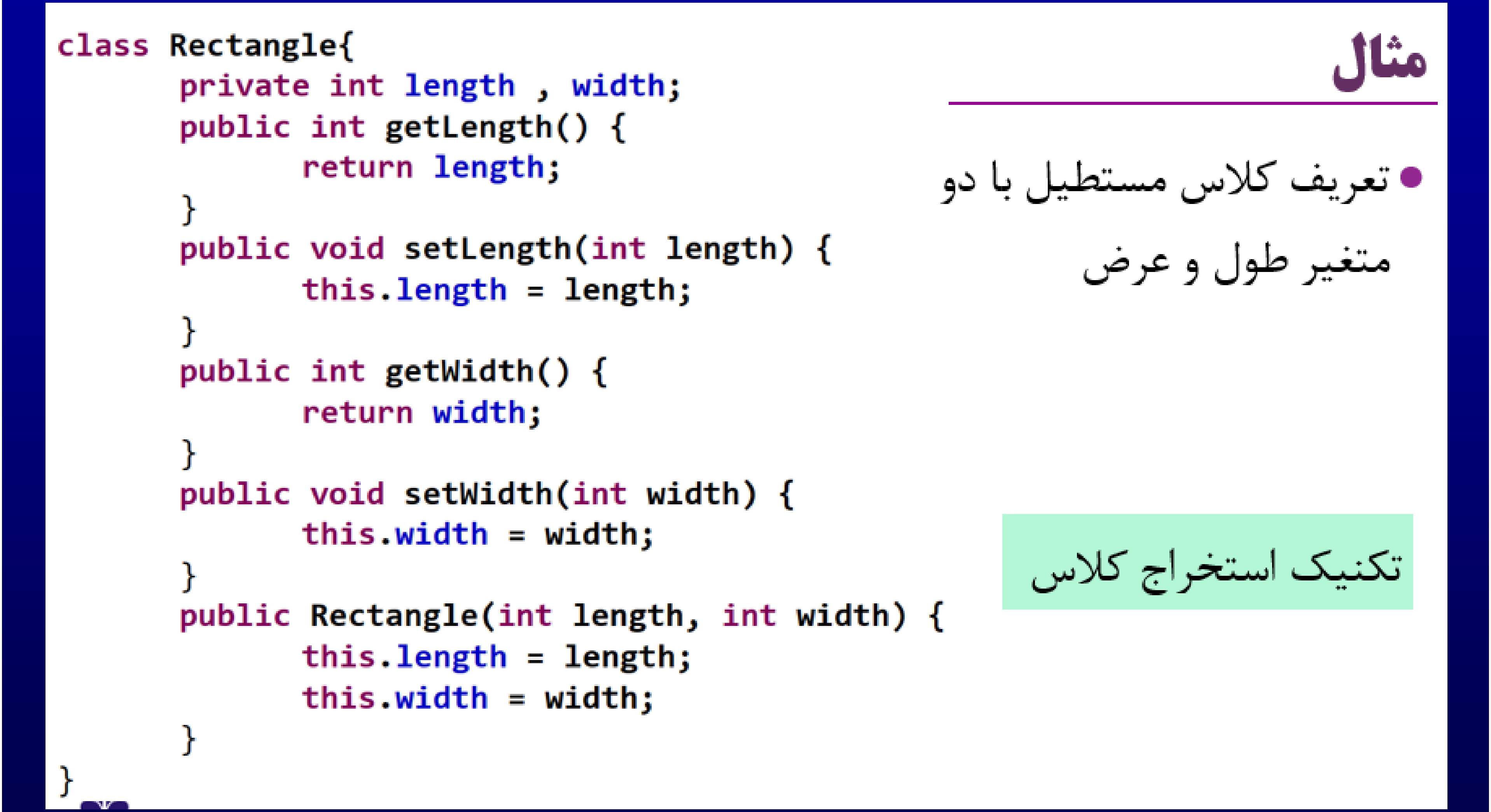
مثال

```
Scanner scanner = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width1 = scanner.nextInt();
System.out.print("Enter the length: ");
int length1 = scanner.nextInt();

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width2 = scanner.nextInt();
System.out.print("Enter the length: ");
int length2 = scanner.nextInt();

int area1 = width1*length1;
int area2 = width2*length2;

if(area1 == area2)
System.out.println("Equal");
```

۲- دسته داده‌ها

(تکرار **گروهی** از متغیرها

در نقاط مختلف کد)

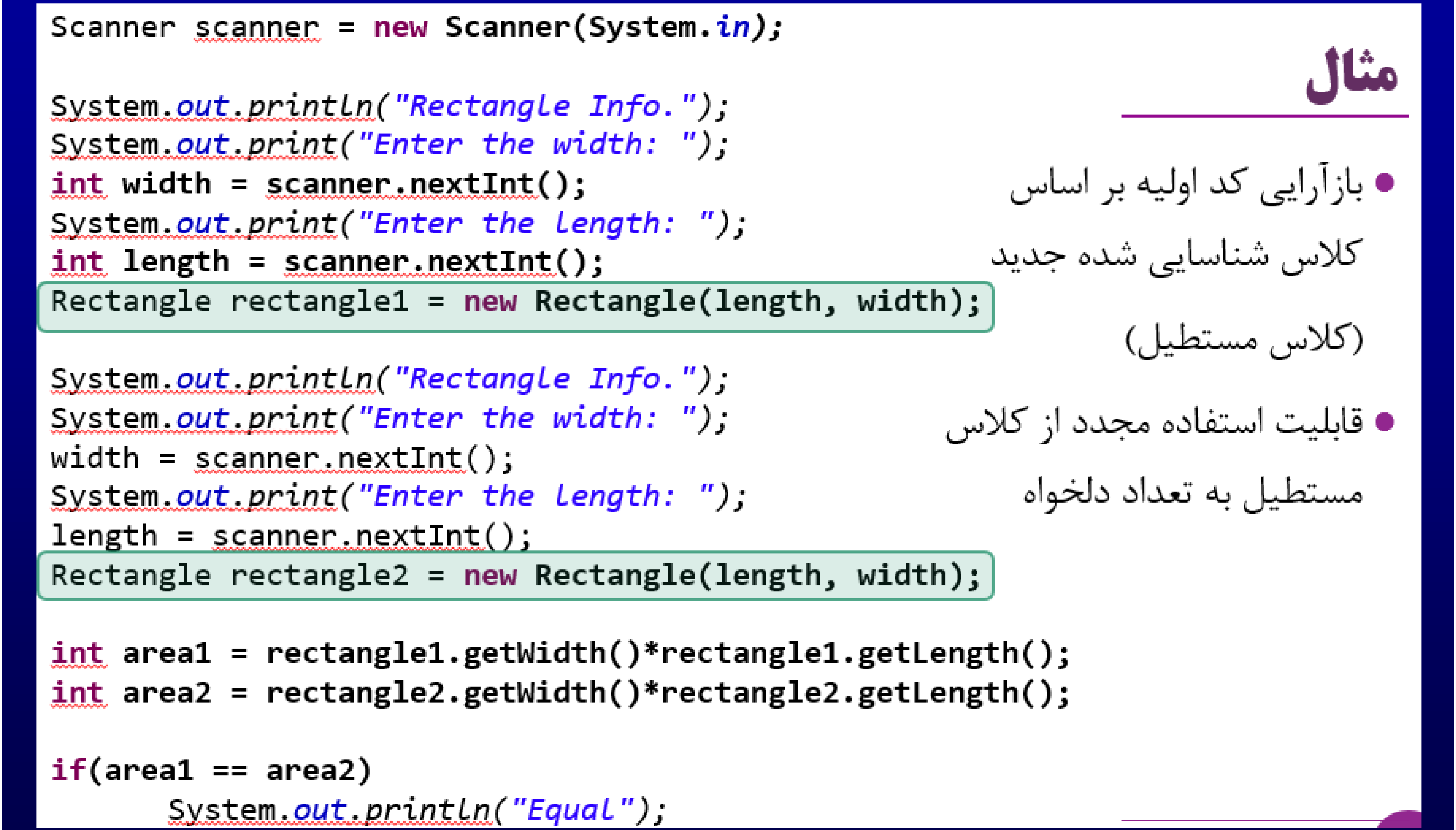


مثال

- تعریف کلاس مستطیل با دو متغیر طول و عرض

تکنیک استخراج کلاس

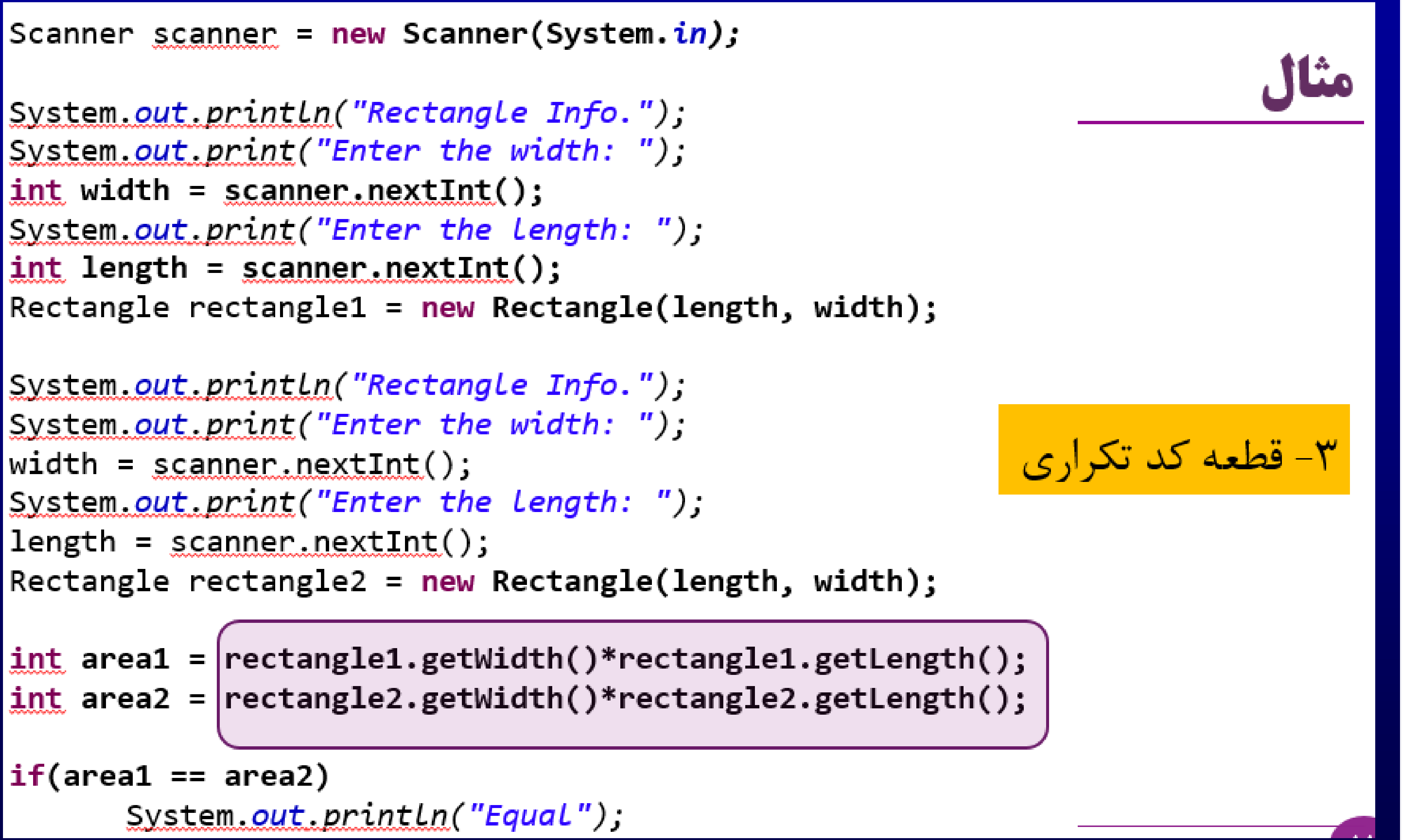
```
class Rectangle{
      private int length , width;
      public int getLength() {
            return length;
      }
      public void setLength(int length) {
            this.length = length;
      }
      public int getWidth() {
            return width;
      }
      public void setWidth(int width) {
            this.width = width;
      }
      public Rectangle(int length, int width) {
            this.length = length;
            this.width = width;
      }
}
```



مثال

- بازآرایی کد اولیه بر اساس کلاس شناسایی شده جدید (کلاس مستطیل)
- قابلیت استفاده مجدد از کلاس مستطیل به تعداد دلخواه

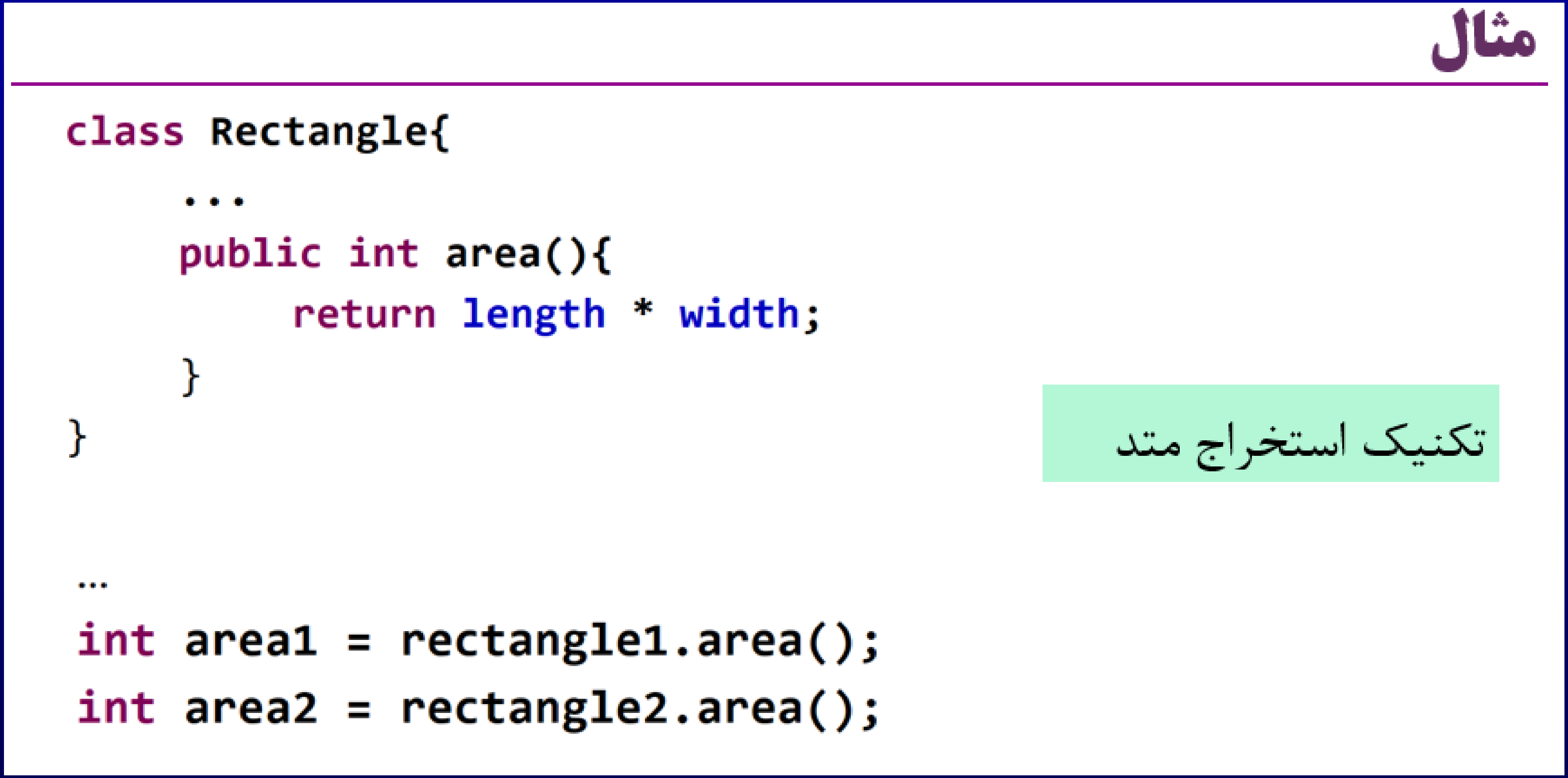
```
Scanner scanner = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width = scanner.nextInt();
System.out.print("Enter the length: ");
int length = scanner.nextInt();
Rectangle rectangle1 = new Rectangle(length, width);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
width = scanner.nextInt();
System.out.print("Enter the length: ");
length = scanner.nextInt();
Rectangle rectangle2 = new Rectangle(length, width);

int area1 = rectangle1.getWidth()*rectangle1.getLength();
int area2 = rectangle2.getWidth()*rectangle2.getLength();

if(area1 == area2)
        System.out.println("Equal");
```



مثال

```
Scanner scanner = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width = scanner.nextInt();
System.out.print("Enter the length: ");
int length = scanner.nextInt();
Rectangle rectangle1 = new Rectangle(length, width);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
width = scanner.nextInt();
System.out.print("Enter the length: ");
length = scanner.nextInt();
Rectangle rectangle2 = new Rectangle(length, width);

int area1 = rectangle1.getWidth()*rectangle1.getLength();
int area2 = rectangle2.getWidth()*rectangle2.getLength();

if(area1 == area2)
      System.out.println("Equal");
```

۳- قطعه کد تکراری

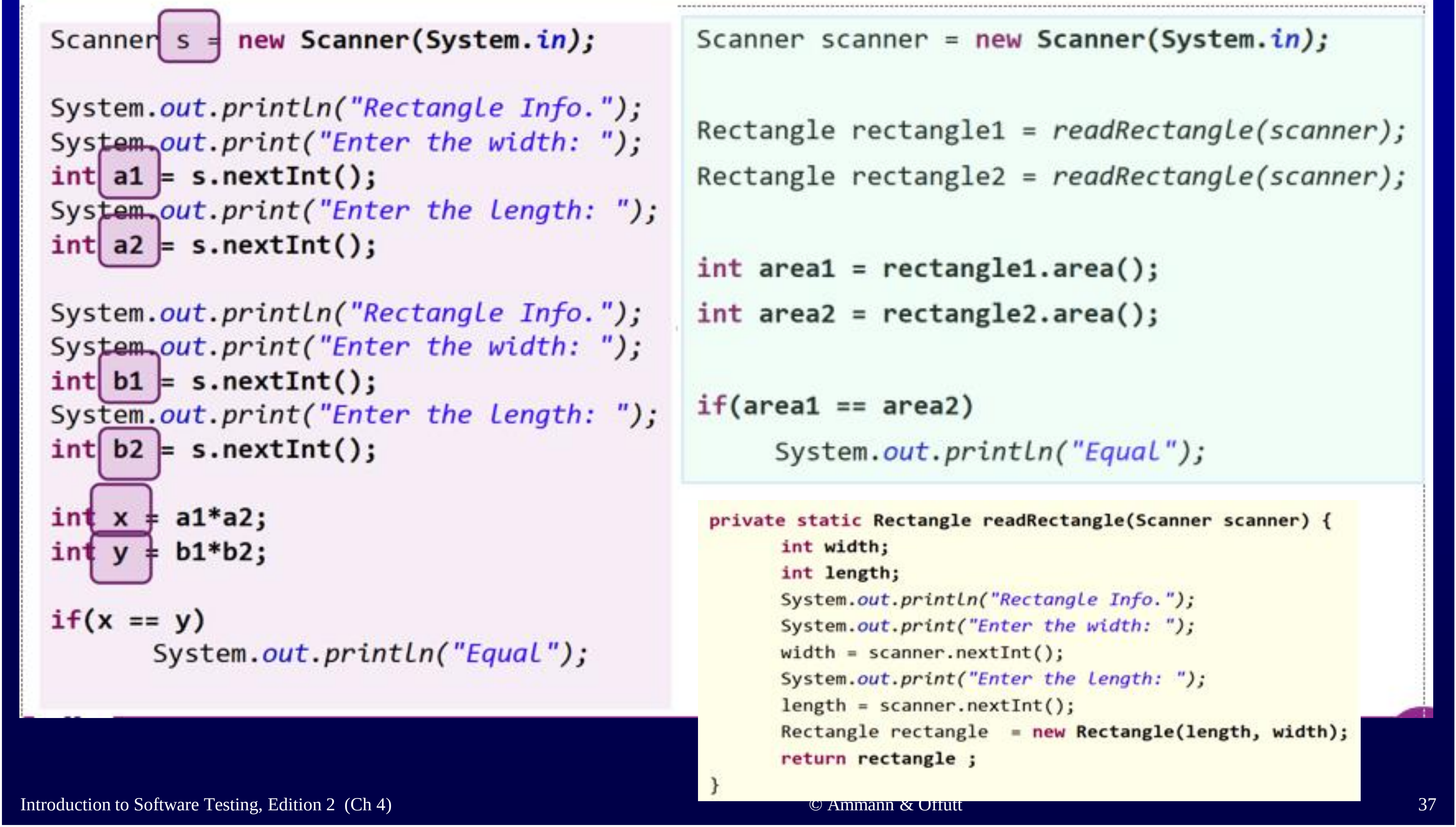


مثال

```
class Rectangle{
    ...
    public int area(){
        return length * width;
    }
}


 ...
 int area1 = rectangle1.area();
 int area2 = rectangle2.area();
```

تکنیک استخراج متد

مقایسه کد اولیه با کد باز آرایی شده



```
Scanner s = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int a1 = s.nextInt();
System.out.print("Enter the length: ");
int a2 = s.nextInt();

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int b1 = s.nextInt();
System.out.print("Enter the length: ");
int b2 = s.nextInt();

int x = a1*a2;
int y = b1*b2;

if(x == y)
        System.out.println("Equal");
```

```
Scanner scanner = new Scanner(System.in);


Rectangle rectangle1 = readRectangle(scanner);
Rectangle rectangle2 = readRectangle(scanner);


int area1 = rectangle1.area();
int area2 = rectangle2.area();


if(area1 == area2)
      System.out.println("Equal");
```

```
private static Rectangle readRectangle(Scanner scanner) {
        int width;
        int length;
        System.out.println("Rectangle Info.");
        System.out.print("Enter the width: ");
        width = scanner.nextInt();
        System.out.print("Enter the length: ");
        length = scanner.nextInt();
        Rectangle rectangle  = new Rectangle(length, width);
        return rectangle ;
}
```