BİL2005
Veri Yapıları

Emre DELİBAŞ

2. Java Temelleri ve OOP

Neden Java?

Özellikleri

- Yaygın kullanım
- 1990'lardan bugüne geliştirilmeye devam ediyor
- Güncel teknolojilere yüksek oranda adaptasyon.
- Program hatalarına karşı hata denetim gücü.

James Gosling

Java ekonomisi

- Mars keşif aracı.
- Cep Telefonları.
- Blu-ray Disk.
- Web Sunucular.
- Medikal cihazlar.
- Supercomputing.
- …

← Milyonlarca gelitirici
Milyarlarca cihaz.

Java dilinin alt kümeleri

Temel Tipler
int
long
double
char
String
boolean

noktalama
{
}
(
)
,
;

Sayısal ifade operasyonları
+
-
*
/
%
++
--

karşılaştırma
<
<=
>
>=
==
!=

String *operasyonları*
+
""
length()
charAt()
compareTo()
matches()

boolean operasyonlar
true
false
!
&&
||

atama
=

flow control
if
else
for
while

diziler
a[]
length
new

Nesne Tabanlı
static
class
public
private
new
final
toString()
main()

type dönüşüm metodları
Integer.parseInt()
Double.parseDouble()

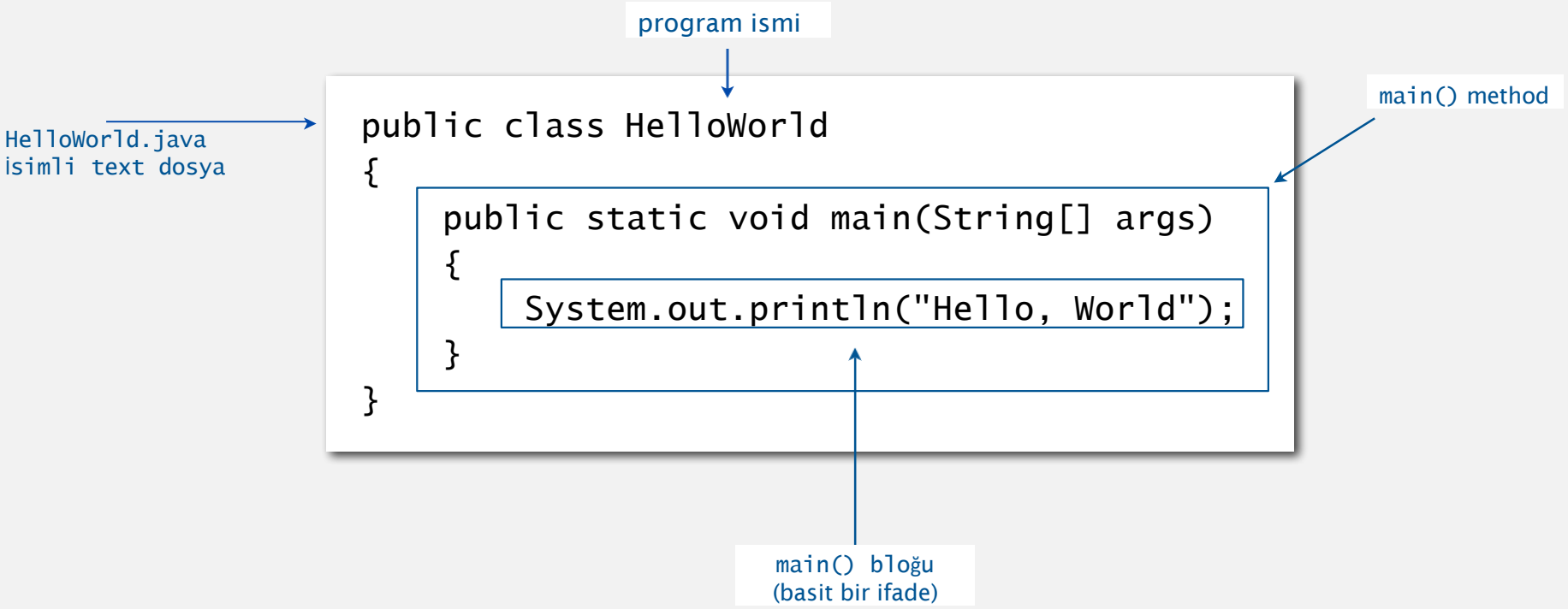
Math *methodları*
Math.sin()
Math.cos()
Math.log()
Math.exp()
Math.pow()
Math.sqrt()
Math.min()
Math.max()
Math.abs()
Math.PI

System *methodları*
System.print()
System.println()
System.printf()

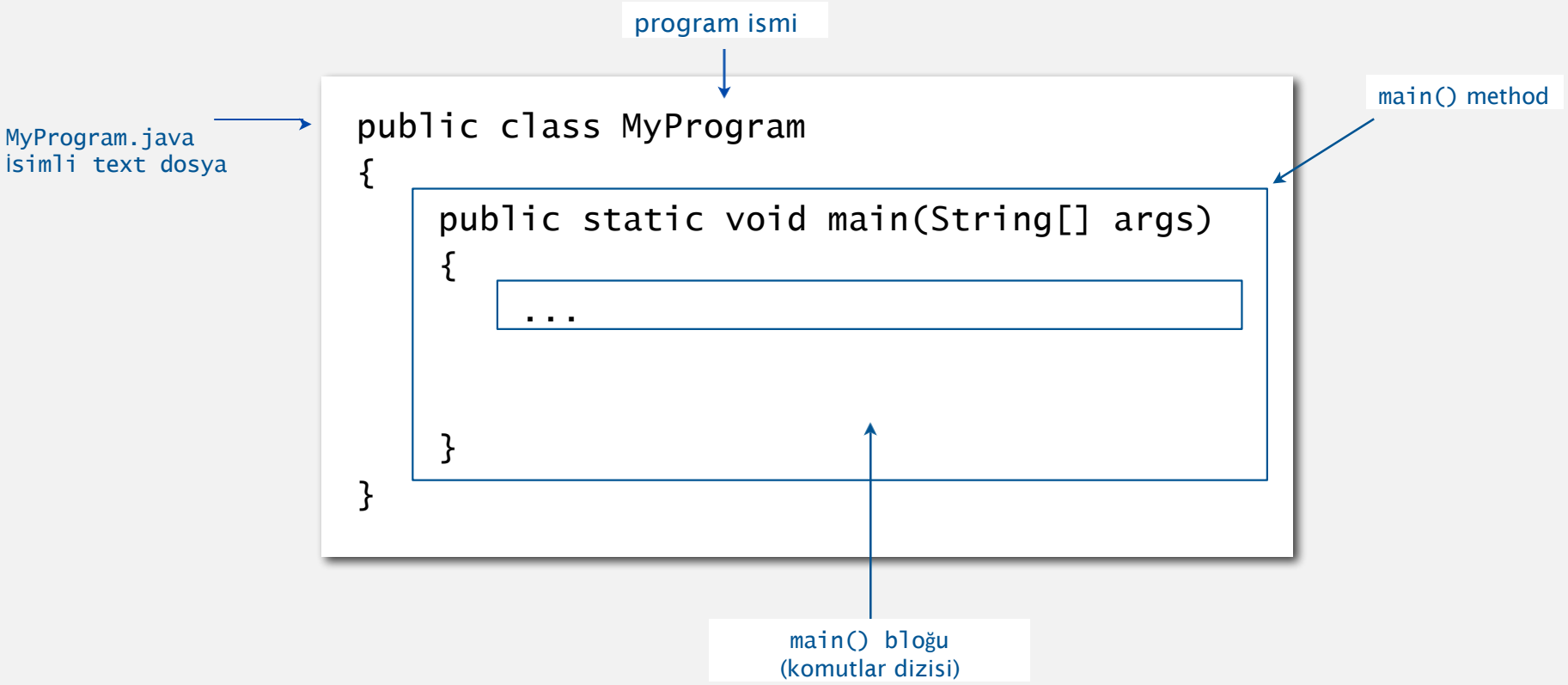
our Std methods
StdIn.read*()
StdOut.print*()
StdDraw.*()
StdAudio.*()
StdRandom.*()

Your programs will primarily consist of these plus identifiers (names) that you make up.

İlk programınızın anatomisi



Diğer programlarınızın anatomisi



Aynı programın üç versiyonu

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World");
    }
}
```

```
/*************************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 *************************************************************************/

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```


```
public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } }
```

Sonuç: Fontlar, renk, yorumlar, ve extra boşluklar Java diliyle alakalı değildir..

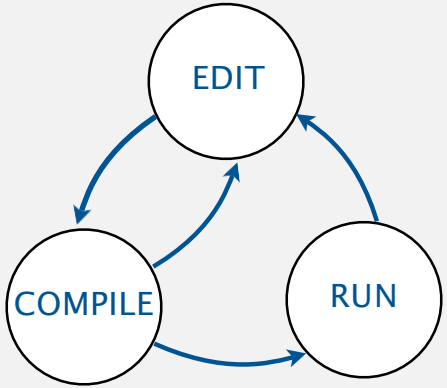
Java da program geliştirme

Üç adımlı bir işlemdir, *feedback içerir.*

1. Programını **YAZ**
   - Bilgisayar klavyesi ile kodları gir.
   - Sonuç: `HelloWorld.java` benzeri bir dosya oluşur.

2. Çalıştırılabilir bir dosya elde etmek için programını **DERLE**
   - Java Compiler kullan
   - Sonuç: Bir Java bytecode dosyası `HelloWorld.class`
   - Hata? 1. Adıma dön düzelt ve tekrar derle.

3. Programını **ÇALIŞTIR**
   - Java runtime kullan.
   - Sonuç: Programının çıktısı.
   - Hata? 1.Adıma git düzelt, yeniden derle ve çalıştır



Temel veri tipleri

Koşul ifadeleri ve döngüler

Diziler

Metotlar

Java ile Yaz, Derle, Çalıştır

Temel Veri tipleri

Bir veri tipi birtakım değerler ve bu değerler üzerindeki operasyonlardır.

Tip	*Alabildiği değerler*	*Örnek*	*işlem örnekleri*
`char`	Karakterler	`'A'` `'@'`	karşılaştırma
`String`	Karakter Dizisi	`"Hello World"` `"CS is fun"`	birleştirme
`int`	Tam Sayılar	`17` `12345`	Topla, çıkar, çarp, böl
`double`	Ondalıklı sayılar	`3.1415` `6.022e23`	Topla, çıkar, çarp, böl
`boolean`	D, Y	`true` `false`	and, or, not

Java'nın Temel Veri Tipleri

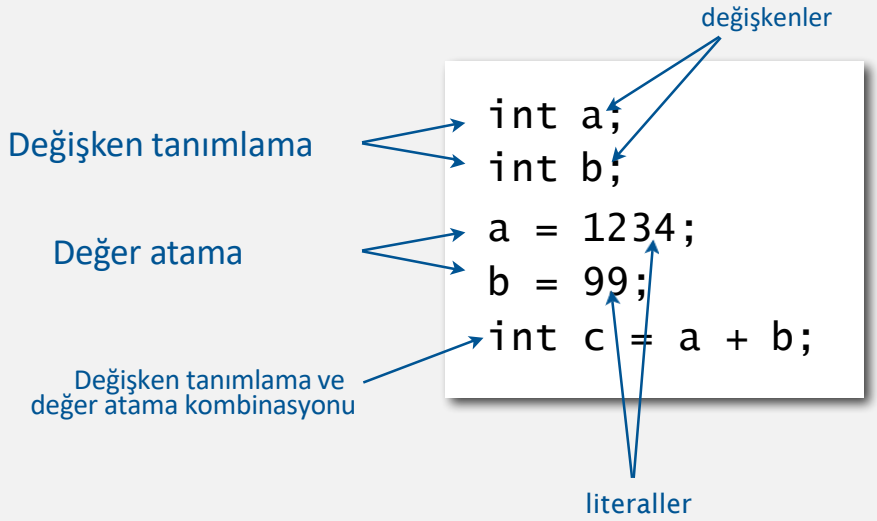
Temel Tanımlamalar

Değişken (variable) bir değere karşılık gelen referanstır.

Literal bir değerin programlama dili gösterimidir.

Değişken Tanımlama bir tip ile değeri ilişkilendirmedir.

Değer atama bir değer ile değişkeni ilişkilendirmedir.



Değişkenler, literaller, atama ve tanımlama. Örnek: değerler arası yer değiştirme

```
public class Exchange
{
    public static void main(String[] args)
    {
        int a = 1234;
        int b = 99;
        int t = a;
        a = b;
        b = t;
    }
}
```

Bu kod a ve b değerlerini yer değiştirir

A trace is a table of variable values after each statement.

	a	b	t
	undeclared	*undeclared*	*undeclared*
`int a = 1234;`	1234	*undeclared*	*undeclared*
`int b = 99;`	1234	99	*undeclared*
`int t = a;`	1234	99	1234
`a = b;`	99	99	1234
`b = t;`	99	1234	1234

Metinlerle işlem yapmak için veri tipi: String

`String` veri tipi

Değerler	Karakter dizileri
Tipik literaller	`"Hello, "  "1 "  " * "`
İşlem	birleştirme
operator	`+`

String işlemleri (concatenation)

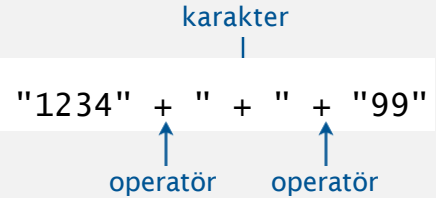
expression	*value*
`"Hi, " + "Bob"`	`"Hi, Bob"`
`"1" + " 2 " + "1"`	`"1 2 1"`
`"1234" + " + " + "99"`	`"1234 + 99"`
`"1234" + "99"`	`"123499"`

Tipik Kullanlımı: Giriş/Çıkış İşlemleri

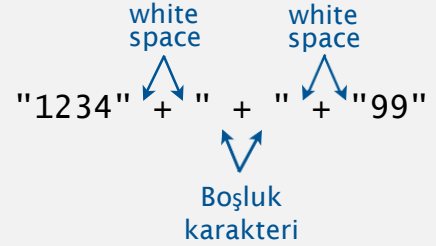
Önemli Not:

Karakterlerin yorumlanması içeriğe bağlıdır.

Ör 1: artı işareti



Ör 2: bosluklar



String işlemleri Örneği: Cetvelin alt bölümleri

```
public class Ruler
{
   public static void main(String[] args)
   {
      String ruler1 = "1";
      String ruler2 = ruler1 + " 2 " + ruler1;
      String ruler3 = ruler2 + " 3 " + ruler2;
      String ruler4 = ruler3 + " 4 " + ruler3;
      System.out.println(ruler4);
   }
}
```

all + ops are concatenation


```
% java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

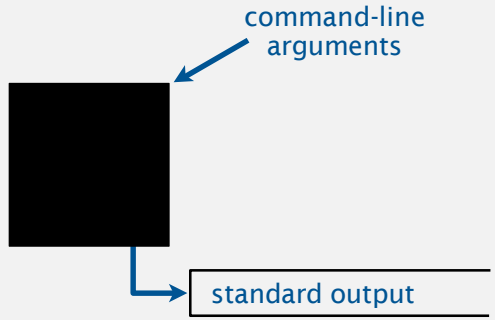
	ruler1	ruler2	ruler3	ruler4
	undeclared	*undeclared*	*undeclared*	*undeclared*
`ruler1 = "1";`	1	*undeclared*	*undeclared*	*undeclared*
`ruler2 = ruler1 + " 2 " + ruler1;`	1	1 2 1	*undeclared*	*undeclared*
`ruler3 = ruler2 + " 3 " + ruler2;`	1	1 2 1	1 2 1 3 1 2 1	*undeclared*
`ruler4 = ruler3 + " 4 " + ruler3;`				1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

Giriş / Çıkış

Programımıza işlenmek üzere veri sunmamız ve işlenen verilerin sonuçlarını alabilmemizi sağlarlar

Kullanıcılar metinlerle çalışmayı tercih ederler.

Programlar sayılarla daha etkili çalışırlar.



Bird's eye view of a Java program

Çıkış

- `System.out.println()` verilen metni yazdıran metot.
- Java sayıları çıktı vermek için otomatik olarak metne çevirir.

Komut satırı Girdisi

- Çalıştırma sırasında programa verilen `args[0]`, `args[1]`, ... şeklindeki string girdilerdir.
- S. *integer* bir değeri komut satırı girdisi olarak nasıl alabiliriz?
- C. String ifadeleri tam sayıya çevirmek için kullanılan `Integer.parseInt()` sistem metodunu çağırırız.

Giriş / çıkış alıştırma: değerleri yer değiştirme

```
public class Exchange
{
    public static void main(String[] args)
    {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int t = a;
        a = b;
        b = t;
        System.out.println(a);
        System.out.println(b);
    }
}
```

Java çıktı işlemi için integer değeri otomatik olarak String e çevirir.

```
% java Exchange 5 2
2
5

% java Exchange 1234 99
99
1234
```

S. Bu program ne yapar?

A. Komut satırından iki değeri okur,daha sonra içeriklerini birbiri arasında değiştirir.

Tam sayı işlemleri için veri tipi: `int`

`int` veri tipi

Değerler	integers between -2^{31} and $2^{31}-1$				
Tipik literaller	1234	99	0	1000000	
Operasyonlar	topla	çıkar	çarp	böl	mod
operatör	+	-	*	/	%

Önemli Not:

Yalnızca 2^{32} farklı int değer.

`int` operasyon örnekleri

ifade	*değer*	*Açıklama*
5 + 3	8	
5 - 3	2	
5 * 3	15	
5 / 3	1	*Tam kısmını alır*
5 % 3	2	*Mod alır*
1 / 0		*runtime error*

Öncelik

ifade	*değer*	*Açıklama*
3 * 5 - 2	13	** öncelik sahibi*
3 + 5 / 2	5	*/ öncelik sahibi*
3 - 5 - 2	-4	*Soldan sıralı*
(3 - 5) - 2	-4	*Tercih edilen*

Tipik kullanım: Matematik hesaplamaları

Int ve String ifadeler arasında tip dönüşümü ile işlemler

```
public class IntOps
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum = a + b;
      int prod = a * b;
      int quot = a / b;
      int rem  = a % b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      System.out.println(a + " / " + b + " = " + quot);
      System.out.println(a + " % " + b + " = " + rem);
   }

}
```

Java String birleştirme işlemi için integer değeri otomatik olarak String e çevirir.

```
% java IntOps 5 2
5 + 2 = 7
5 * 2 = 10
5 / 2 = 2
5 % 2 = 1

% java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
```

Ondalıklı sayılarla işlem yapabilmek için: double

double veri tipi

değerler	real numbers				
tipik literaller	3.14159	2.0	1.4142135623730951	6.022e23	
operasyonlar	topla	çıkar	çarp	böl	mod
operatörler	+	-	*	/	%

6.022 10^{23}

Tipik double değerler yaklaşık değerlerdir

Örnek:
π için double değer yoktur.
$2^{1/2}$ için double değer yoktur.
1/3 için double değer yoktur.

double operasyon örnekleri

ifade	*değer*
`3.141 + .03`	`3.171`
`3.141 - .03`	`3.111`
`6.02e23/2`	`3.01e23`
`5.0 / 3.0`	`1.6666666666666667`
`10.0 % 3.141`	`0.577`
`Math.sqrt(2.0)`	`1.4142135623730951`

Özel değerler

ifade	*değer*
`1.0 / 0.0`	`Infinity`
`Math.sqrt(-1.0)`	`NaN`

"not a number"

Diğer sayısal veri tipleri

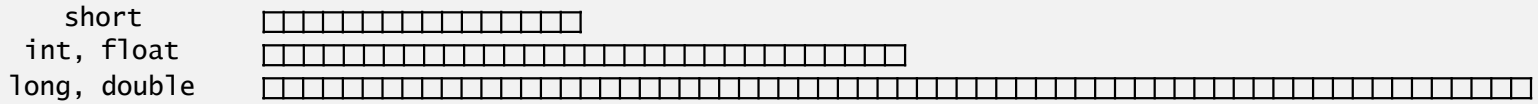
`short` veri tipi	
değerler	integers between -2^{15} and $2^{15}-1$
operasyonlar	[same as `int`]

`long` veri tipi	
değerler	-2^{63} and $2^{63}-1$ arasındaki tam say.
operasyonlar	[`int` ile aynı]

`float` veri tipi	
değerler	real sayılar
operasyonlar	[`double` ile aynı]

Why different numeric types?

- Bellek kullanımı ve tam sayı aralığı arasındaki tercihler.
- Real sayılardaki bellek kullanımı ve hassasiyet tercihleri.



Java'nın Math kütüphanesinden örnekler

public class Math	
double abs(double a)	*Mutlak değer*
double max(double a, double b)	*Max a ve b*
double min(double a, double b)	*Min a ve b*
double sin(double theta)	*Sin fonksiyonu*
double cos(double theta)	*cos fonksiyonu*
double tan(double theta)	*tan fonksiyonu*
double exp(double a)	*(e^a)*
double log(double a)	*doğal log ($\log_e$ a, or ln a)*
double pow(double a, double b)	*(a^b)*
long round(double a)	*En yakın tamsayıya yuvarla*
double random()	*[0. 1) arası random sayı*
double sqrt(double a)	*karekök* a
double E	*e (constant)*
double PI	*π (constant)*

int, long, ve float için de tanımlanabilir

asin(), acos(), and atan() ters fonksiyonları da mevcuttur.

Radian olarak alır. toDegrees() ve toRadians() dönüşüm de kullanılabilir

Artık hesap makinesini bırakabilirsin. (lütfen!).

Ondalıklı sayılar örneği: ikinci derece denklemler

Cebir örneği: $x^2 + bx + c$ için kökler $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

```
public class Quadratic
{
   public static void main(String[] args)
   {

      // Parse coefficients from command-line.
      double b = Double.parseDouble(args[0]);
      double c = Double.parseDouble(args[1]);

      // Calculate roots of x*x + b*x + c.
      double discriminant = b*b - 4.0*c;
      double d = Math.sqrt(discriminant);
      double root1 = (-b + d) / 2.0;
      double root2 = (-b - d) / 2.0;

      // Print them out.
      System.out.println(root1);
      System.out.println(root2);
   }
}
```

```
% java Quadratic -3.0 2.0
2.0                                   x^2 − 3x + 2
1.0

% java Quadratic -1.0 -1.0
1.618033988749895                     x^2 − x − 1
-0.6180339887498949

% java Quadratic 1.0 1.0
NaN                                   x^2 + x + 1
NaN

% java Quadratic 1.0 hello
java.lang.NumberFormatException: hello

% java Quadratic 1.0
java.lang.ArrayIndexOutOfBoundsException
```

İki argüman gerekli.
(Hayatın gerçekleri: Tüm hatalar anlaşılır değildir.)

true / false verilerde işlemler: boolean

boolean veri tipi

değerler	true	false	
literaller	true	false	
operasyonlar	and	or	not
operatörler	&&	||	!

Doğruluk tablosu tanımları

a	!a		a	b	a && b	a || b
true	false		false	false	false	false
false	true		false	true	false	true
			true	false	false	true
			true	true	true	true

S. a XOR b?

C. (!a && b) || (a && !b)

İspat

a	b	!a && b	a && !b	(!a && b) || (a && !b)
false	false	false	false	false
false	true	true	false	true
true	false	false	true	true
true	true	false	false	false

Karşılaştırma Operatörleri

Temel veri tipleri arasında karşılaştırma yapmak için kullanılır.

- Operand: Aynı veri tipindeki iki değer.
- Sonuç: Boolean bir değer.

operatör	*Anlamı*	true	false
==	Eşit	2 == 2	2 == 3
!=	Eşit değil	3 != 2	2 != 2
<	Küçük	2 < 13	2 < 2
<=	Büyük eşit	2 <= 2	3 <= 2
>	Büyük	13 > 2	2 < 13
>=	Büyük eşit	3 >= 2	2 >= 3

Örnekler

non-negative discriminant?	`( b*b - 4.0*a*c ) >= 0.0`
Yüzyılın başlangıcı?	`( year % 100 ) == 0`
Doğru ay değeri?	`( month >= 1 ) && ( month <= 12 )`

Tipik double değerler *yaklaşık değer* olduğundan == karşılaştırmalarında dikkatli olun

Boolean ile hesaplama örneği: artık yıl kontrolü

Q. Girilen yıl artık yıl mıdır?

A. Eğer (i) 400 e bölünebilen veya (ii) 4 e bölünebilen fakat 100 e bölünmeyen

```
public class LeapYear
{
   public static void main(String[] args)
   {
      int year = Integer.parseInt(args[0]);
      boolean isLeapYear;

      // divisible by 4 but not 100
      isLeapYear = (year % 4 == 0) && (year % 100 != 0);

      // or divisible by 400
      isLeapYear = isLeapYear || (year % 400 == 0);

      System.out.println(isLeapYear);
   }
}
```

```
% java LeapYear 2016
true

% java LeapYear 1993
false

% java LeapYear 1900
false

% java LeapYear 2000
true
```

Tip kontrol

Değişken tipleri daima operasyonlardaki tanımlarla eşleşmelidir.

Java compiler sizin dostunuzdur: kodunuzdaki tip hatalarını kontrol eder.

```
public class BadCode
{
   public static void main(String[] args)
   {
      String s = "123" * 2;
   }
}
```

```
% javac BadCode.java
BadCode.java:5: operator * cannot be applied to java.lang.String,int
        String s = "123" * 2;
                           ^
1 error
```


Uygun olduğunda, sık sık bir tipteki değeri başka bir tipe, tip uyumunu sağlamak için dönüştürürüz

Temel veri tiplerinde tip dönüşümü

Tip dönüşümleri programlamanın doğasında olan işlemlerdir.

Otomatik

- "+" için sayıları stringe dönüştürme.
- Hassasiyet kaybı yok ise sayısal tiplerin dönüşümü.

ifade	*tip*	*değer*
`"x: " + 99`	`String`	`"x: 99"`
`11 * 0.25`	`double`	`2.75`

Fonksiyon parametrelerinde açıkça tanımlı durumlar

`Integer.parseInt("123")`	`int`	`123`
`Math.round(2.71828)`	`long`	`3`

`(int) 2.71828`	`int`	`2`
`(int) Math.round(2.71828)`	`int`	`3`
`11 * (int) 0.25`	`int`	`0`

Verinizin tipine dikkat edin!

Tip dönüşümleri mantıksız gelebilir ancak uygulamada oldukça pratik çözümler sunar

Tip dönüşümü Quiz

S. Aşağıdaki ifadelerin her birinin tip ve sonuçlarını belirtiniz.

a. `( 7 / 2 ) * 2.0`

b. `( 7 / 2.0 ) * 2`

c. `"2" + 2`

d. `2.0 + "2"`

Tip dönüşümü Quiz

S. Aşağıdaki ifadelerin her birinin tip ve sonuçlarını belirtiniz.

a. `( 7 / 2 ) * 2.0` 6.0, `double` (7/2 is 3, an `int`)

b. `( 7 / 2.0 ) * 2` 7.0, `double`

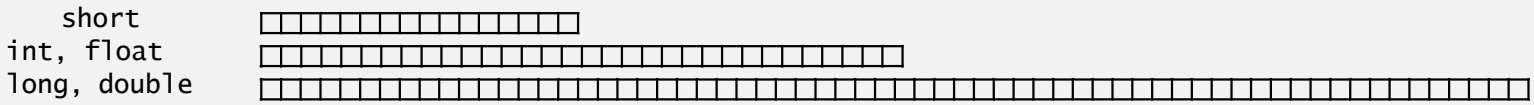
c. `"2" + 2` 22, `String`

d. `2.0 + "2"` 2.02, `String`

Tip dönüşümlerinin hikayesi

Neden farklı nümerik tipler?

- Bellek kullanımı ve ihtiyaç duyulan sayı aralığı konusundaki tercihler.
- Bellek kullanımı ve hassasiyet ihtiyacı konusundaki tercihler.



Bir dönüşümün imkansız olduğu durumlar..

- Ör: (short) 70000.
- Short değerler -2^{15} and $2^{15}-1 = 32767$.

Tip dönüşümünün kullanışlı olduğu durumlar: Random integer lar

Math.Random() sistem metodu [0,1) aralığında rastgele sayılar üretir.

Problem: *N* sayısı girilsin, 0 ile N-1 arasında bir random tam sayı üretin.

```
public class RandomInt
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      double r = Math.random();
      int t = (int) (r * N);
      System.out.println(t);
   }
}
```

String to int (system method)

double to int (cast)

int to double (automatic)

```
% java RandomInt 6
3

% java RandomInt 6
0

% java RandomInt 10000
3184
```

Bir veri tipi birtakım değerler ve bu değerler üzerindeki operasyonlardır.

Java'da sık kullanılan temel veri tipleri

- String, giriş çıkışta kullanılan karakterler dizisidir.
- int, tam sayılarla hesaplama ve programlardaki matematik hesaplamalar için kullanılır.
- double, ondalıklı sayılarda hesaplamalar işlemler için kullanılır.
- boolean, İki durumlu işler için kullanılır, programlardaki karar verme yapıları için kullanılır.

Temel veri tipleri

Koşul ifadeleri ve döngüler

Diziler

Metotlar

Java ile Yaz, Derle, Çalıştır

Context: basic building blocks for programming

any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math | text I/O

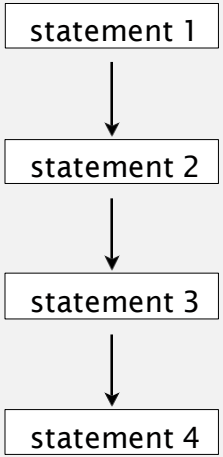
primitive data types | assignment statements

Previous lecture:
equivalent to a calculator

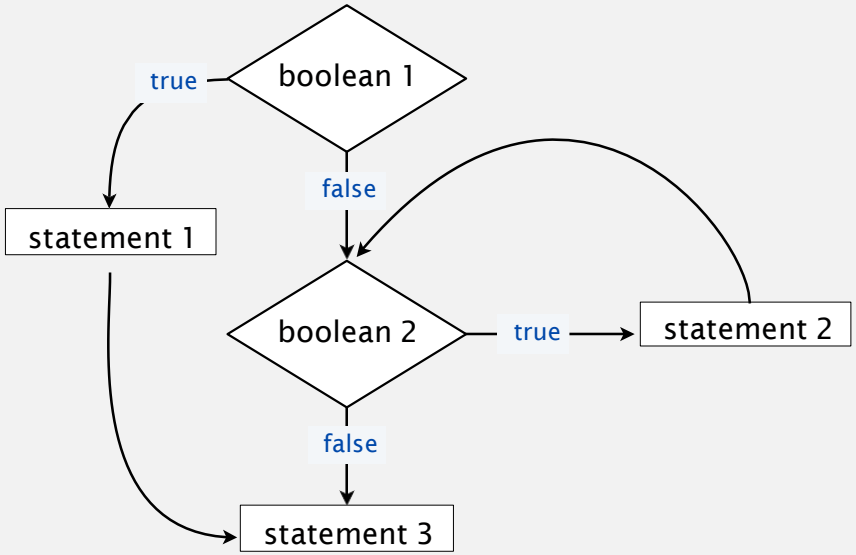
This lecture:
to infinity and beyond!

Control flow

- The sequence of statements that are actually executed in a program.
- Conditionals and loops enable us to choreograph control flow.



straight-line control flow
[previous lecture]



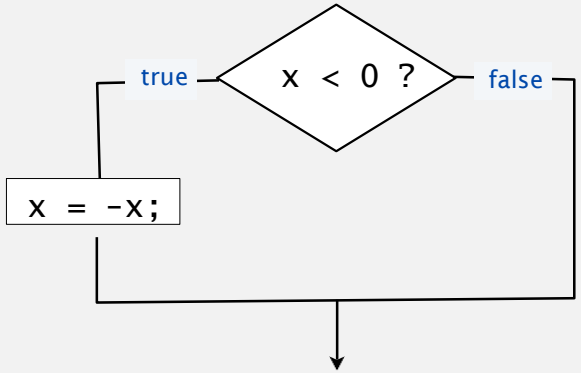
control flow with conditionals and a loop
[this lecture]

The `if` statement

Execute certain statements depending on the values of certain variables.

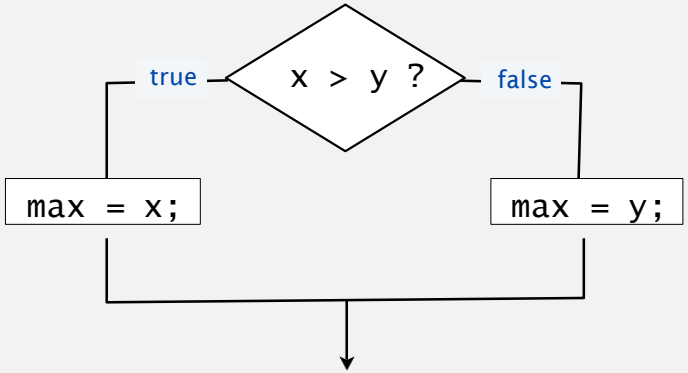
- Evaluate a `boolean` expression.
- If `true`, execute a statement.
- The `else` option: If `false`, execute a different statement.

Example: `if (x < 0) x = -x;`



Replaces x with the absolute value of x

Example:
```
if (x > y) max = x;
else       max = y;
```



Computes the maximum of x and y

Example of if statement use: simulate a coin flip

```
public class Flip
{
   public static void main(String[] args)
   {
      if (Math.random() < 0.5)
           System.out.println("Heads");
      else
           System.out.println("Tails");
   }
}
```

```
% java Flip
Heads

% java Flip
Heads

% java Flip
Tails

% java Flip
Heads
```

Example of `if` statement use: 2-sort

Q. What does this program do?

```
public class TwoSort
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      if (b < a)
      {
         int t = a;
         a = b;
         b = t;
      }
      System.out.println(a);
      System.out.println(b);
   }
}
```

alternatives for if and else can be a *sequence* of statements, enclosed in braces

```
% java TwoSort 1234 99
99
1234

% java TwoSort 99 1234
99
1234
```

A. Reads two integers from the command line, then prints them out in numerical order.

Pop quiz on `if` statements

Q. Add code to this program that puts a, b, and c in numerical order.

```
public class ThreeSort
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int c = Integer.parseInt(args[2]);

      System.out.println(a);
      System.out.println(b);
      System.out.println(c);
   }
}
```

```
% java ThreeSort 1234 99 1
1
99
1234

% java ThreeSort 99 1 1234
1
99
1234
```

Pop quiz on `if` statements

Q. Add code to this program that puts a, b, and c in numerical order.

A.

```
public class ThreeSort
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int c = Integer.parseInt(args[2]);
      if (b < a)
      { int t = a; a = b; b = t; }
      if (c < a)
      { int t = a; a = c; c = t; }
      if (c < b)
      { int t = b; b = c; c = t; }
      System.out.println(a);
      System.out.println(b);
      System.out.println(c);
   }
}
```

makes *a* smaller than *b*

makes *a* smaller than both *b* and *c*

makes *b* smaller than *c*

```
% java ThreeSort 1234 99 1
1
99
1234

% java ThreeSort 99 1 1234
1
99
1234
```

Example of `if` statement use: error checks

```
public class IntOps
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum = a + b;
      int prod = a * b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);


      if (b == 0) System.out.println("Division by zero");
      else        System.out.println(a + " % " + b + " = " + a % b);
   }
}
```

```
% java IntOps 5 2
5 + 2 = 7
5 * 2 = 10
5 / 2 = 2
5 % 2 = 1

% java IntOps 5 0
5 + 0 = 5
5 * 0 = 0
Division by zero
Division by zero
```


Good programming practice. Use conditionals to check for *and avoid* runtime errors.

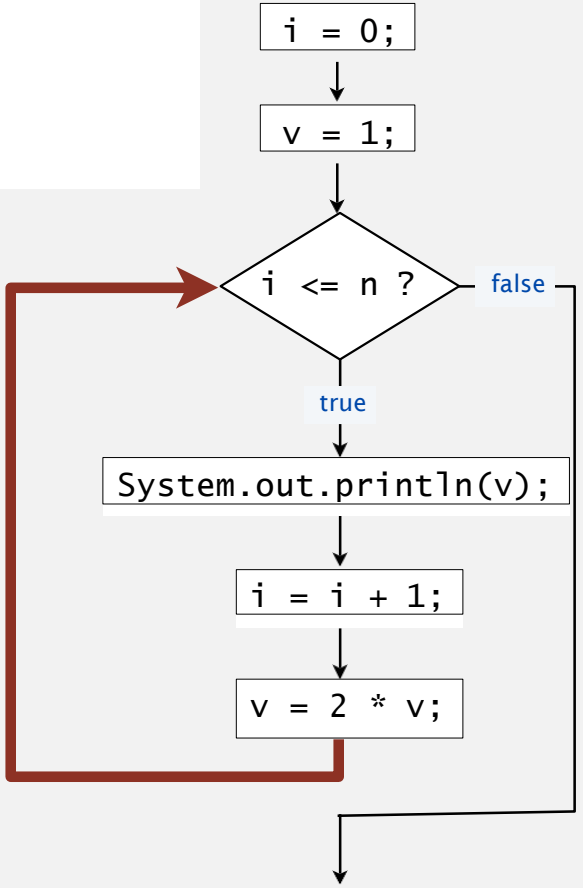
The `while` loop

Execute certain statements repeatedly until certain conditions are met.

- Evaluate a `boolean` expression.
- If `true`, execute a sequence of statements.
- Repeat.

Example:

```
int i = 0;
int v = 1;
while (i <= n)
{
   System.out.println(v);
   i = i + 1;
   v = 2 * v;
}
```



Prints the powers of two from 2^0 to 2^n.

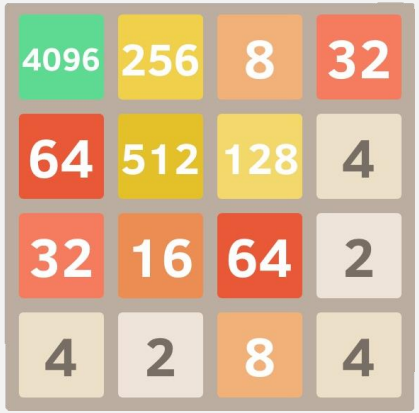
[stay tuned for a trace]

Example of `while` loop use: print powers of two

A trace is a table of variable values after each statement.

```
public class PowersOfTwo
{
   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      int i = 0;
      int v = 1;
      while (i <= n)
      {
         System.out.println(v);
         i = i + 1;
         v = 2 * v;
      }
   }
}
```

i	v	i <= n
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	true
6	64	true
7	128	false




```
% java PowersOfTwo 6
1
2
4
8
16
32
64
```

Prints the powers of two from 2^0 to 2^n.

Pop quiz on `while` loops

Q. Anything wrong with the following code?

```
public class PQwhile
{
   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      int i = 0;
      int v = 1;
      while (i <= n)
         System.out.println(v);
         i = i + 1;
         v = 2 * v;
   }
}
```

Pop quiz on `while` loops

Q. Anything wrong with the following code?

```
public class PQwhile
{
   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      int i = 0;
      int v = 1;
      while (i <= n)
      {  System.out.println(v);
         i = i + 1;
         v = 2 * v;  }
   }
}
```

A. Yes! Needs braces.

Q. What does it do (without the braces)?

A. Goes into an *infinite loop*.

```
% java PQwhile 6
1
1
1
1
1
1
1
1
1
1
```

challenge: figure out how to stop it on your computer

Example of `while` loop use: implement `Math.sqrt()`

Goal. Implement square root function.

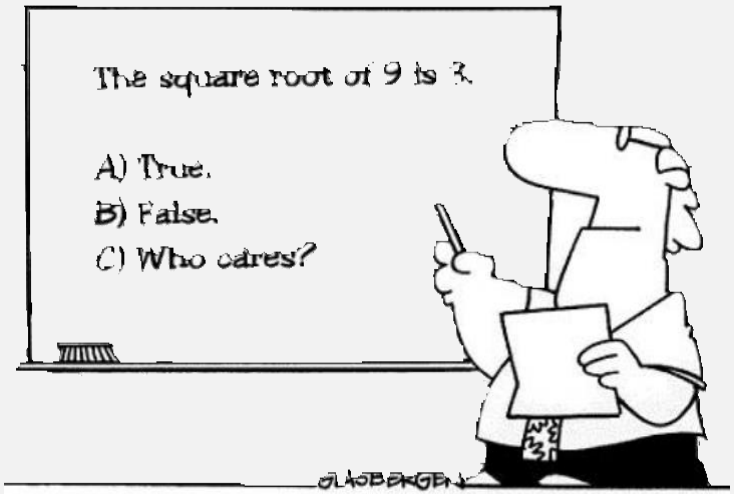
```
% java Sqrt 60481729.0
7777.0
% java Sqrt 2.0
1.4142136
```

Newton-Raphson method to compute $\sqrt{c}$

- Initialize $t_0 = c$.
- Repeat until $t_i = c/t_i$ (up to desired precision): (if $t = c/t$ then $t^2 = c$)
  Set t_{i+1} to be the average of t_i and c / t_i.

i	t_i	$2/t_i$	*average*
0	2	1	1.5
1	1.5	1.3333333	1.4166667
2	1.4166667	1.4117647	1.4142157
3	1.4142157	1.4142114	1.4142136
4	1.4142136	1.4142136	

computing the square root of 2 to seven places



Many students actually look forward to Mr. Atwadder's math tests.

Example of `while` loop use: implement `Math.sqrt()`

Newton-Raphson method to compute $\sqrt{c}$

- Initialize $t_0 = c$.
- Repeat until $t_i = c / t_i$ (up to desired precision):
  Set t_{i+1} to be the average of t_i and c / t_i.

```
public class Sqrt
{
   public static void main(String[] args)
   {
      double EPS = 1E-15;   ← error tolerance (15 places)
      double c = Double.parseDouble(args[0]);
      double t = c;
      while (Math.abs(t - c/t) > t*EPS)
         t = (c/t + t) / 2.0;
      System.out.println(t);
   }
}
```

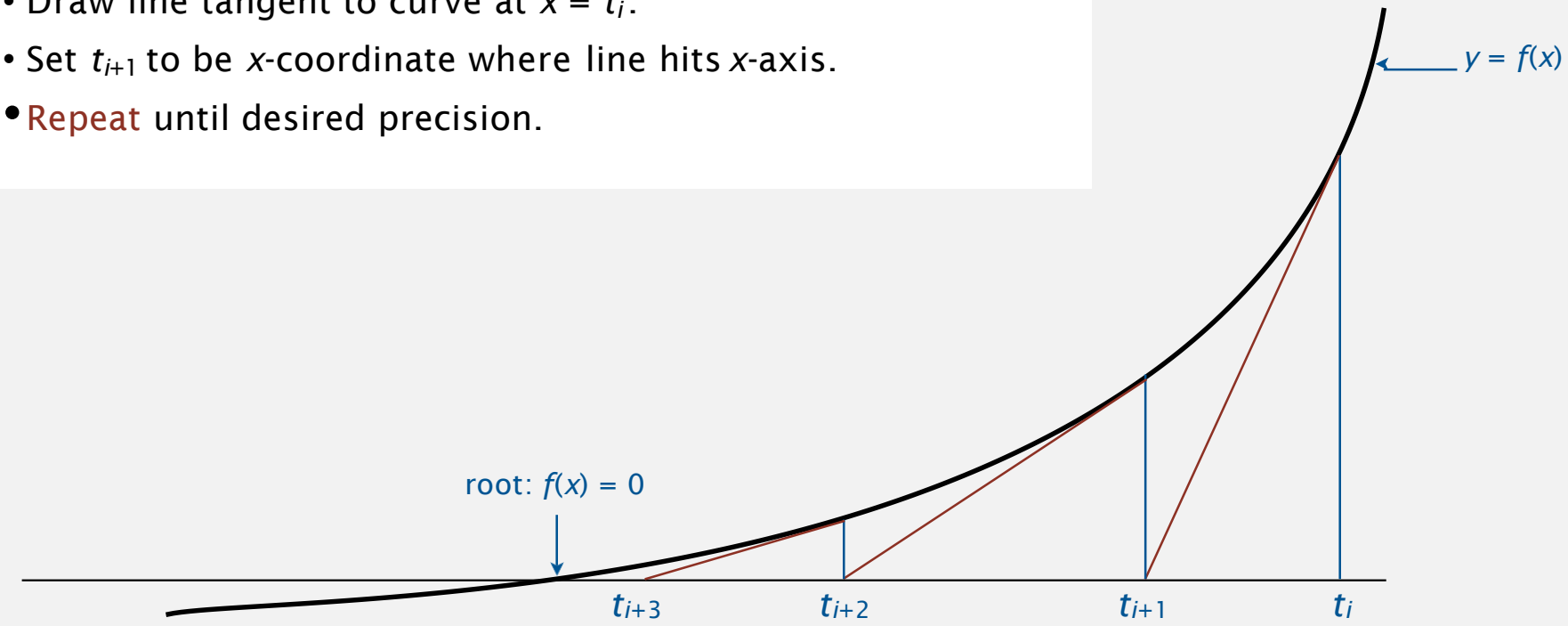
```
% java Sqrt 60481729.0
7777.0

% java Sqrt 2.0
1.414213562373095
```

Newton-Raphson method

Explanation (some math omitted)

- Goal: find *root* of function $f(x)$ (value of x for which $f(x) = 0$). ← use $f(x) = x^2 - c$ for $\sqrt{c}$
- Start with estimate t_0.
- Draw line tangent to curve at $x = t_i$.
- Set t_{i+1} to be x-coordinate where line hits x-axis.
- Repeat until desired precision.



The for loop

An alternative repetition structure. ← Why? Can provide code that is more compact and understandable.

- Evaluate an *initialization statement*.
- Evaluate a *boolean expression*.
- If true, execute a *sequence of statements*, then execute an *increment statement*.
- Repeat.

Example:

```
int v = 1;
for ( int i = 0; i <= n; i++ )
{
   System.out.println( i + " " + v );
   v = 2*v;
}
```

initialization statement

boolean expression

increment statement

Prints the powers of two from 2^0 to 2^n

Every for loop has an equivalent while loop:

```
int v = 1;
int i = 0;
while ( i <= n; )
{
   System.out.println( i + " " + v );
   v = 2*v;
   i++;
}
```

Examples of `for` loop use

```
int sum = 0;
for (int i = 1; i <= N; i++)
   sum += i;
System.out.println(sum);
```

Compute sum (1 + 2 + 3 + . . . + N)

sum	i
1	1
3	2
6	3
10	4

trace at end of loop for $N = 4$

```
long product = 1;
for (int i = 1; i <= N; i++)
   product *= i;
System.out.println(product);
```

Compute N! = 1 * 2 * 3 * . . . * N

product	i
1	1
2	2
6	3
24	4

```
for (int k = 0; k <= N; k++)
   System.out.println(k + " " + 2*Math.PI*k/N);
```

Print a table of function values

k	$\frac{2nk}{U}$
0	0
1	1.57079632...
2	3.14159265...
3	4.71238898...
4	6.28318530...

```
int v = 1;
while (v <= N/2)
   v = 2*v;
System.out.println(v);
```

Print largest power of 2 less than or equal to N

v
2
4
8
16

trace at end of loop for $N = 23$

Example of `for` loop use: subdivisions of a ruler

Create subdivisions of a ruler to $1/N$ inches.

- Initialize `ruler` to one space.
- For each value `i` from `1` to `N`:
  sandwich `i` between two copies of `ruler`.


```
public class Ruler
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      String ruler = " ";
      for (int i = 1; i <= N; i++)
        ruler = ruler + i + ruler;
      System.out.println(ruler);
   }
}
```

i	ruler
1	" 1 "
2	" 1 2 1 "
3	" 1 2 1 3 1 2 1 "
4	" 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 "

End-of-loop trace

```
java Ruler 4
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

```
% java Ruler 100
Exception in thread "main"
java.lang.OutOfMemoryError
```

$2^{100} - 1$ integers in output (!)

Note: Small program can produce huge amount of output.

Pop quiz on for loops

Q. What does the following program print?

```
public class PQfor
{
   public static void main(String[] args)
   {
      int f = 0, g = 1;
      for (int i = 0; i <= 10; i++)
      {
         System.out.println(f);
         f = f + g;
         g = f - g;
      }
   }
}
```

Pop quiz on for loops

Q. What does the following program print?

```
public class PQfor
{
   public static void main(String[] args)
   {
      int f = 0, g = 1;
      for (int i = 0; i <= 10; i++)
      {
         System.out.println(f);
         f = f + g;
         g = f - g;
      }
   }
}
```

A.

Beginning-of-loop trace

i	f	g
0	0	1
1	1	0
2	1	1
3	2	1
4	3	2
5	5	3
6	8	5
7	13	8
8	21	13
9	34	21
10	55	34

values printed

Nesting conditionals and loops

Nesting

- Any "statement" within a conditional or loop may itself be a conditional or a loop statement.
- Enables complex control flows.
- Adds to challenge of debugging.

Example:

```
for (int t = 0; t < trials; t++)
{
   int cash = stake;
   while (cash > 0 && cash < goal)
      if (Math.random() < 0.5) cash++;
      else                     cash--;
   if (cash == goal) wins++;
}
```

`if-else` statement within a `while` loop within a `for` loop

[Stay tuned for an explanation of this code.]

Example of nesting conditionals: Tax rate calculation

Goal. Given income, calculate proper tax rate.

income	rate
0 – $47,450	22%
$47,450 – $114,649	25%
$114,650 – $174,699	28%
$174,700 – $311,949	33%
$311,950 +	35%

```
if (income <  47450) rate = 0.22;
else
   {
      if (income < 114650) rate = 0.25;
      else
         {
            if (income < 174700) rate = 0.28;
            else
               {
                  if (income < 311950) rate = 0.33;
                  else                 rate = 0.35;
               }
         }
   }
```

`if` statement within an `if` statement

`if` statement within an `if` statement within an `if` statement

`if` statement within an `if` statement within an `if` statement within an `if` statement

Pop quiz on nested `if` statements

Q. Anything wrong with the following code?

```
public class PQif
{
   public static void main(String[] args)
   {
      double income = Double.parseDouble(args[0]);
      double rate = 0.35;
      if (income < 47450) rate = 0.22;
      if (income < 114650) rate = 0.25;
      if (income < 174700) rate = 0.28;
      if (income < 311950) rate = 0.33;
      System.out.println(rate);
   }
}
```

Pop quiz on nested `if` statements

Q. Anything wrong with the following code?

```
public class PQif
{
   public static void main(String[] args)
   {
      double income = Double.parseDouble(args[0]);
      double rate = 0.35;
      if (income < 47450) rate = 0.22;
 else if (income < 114650) rate = 0.25;
 else if (income < 174700) rate = 0.28;
 else if (income < 311950) rate = 0.33;
      System.out.println(rate);
   }
}
```

Note. Braces are not needed in this case, but BE CAREFUL when nesting `if-else` statements because of potential ambiguity (see Q&A p. 75).

A. Yes! Need `else` clauses. Without them, code is equivalent to:

```
if (income < 311950) rate = 0.33;
else                 rate = 0.35;
```

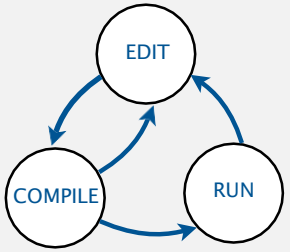
Debugging

is 99% of program development in any programming language, *even for experts.*

Bug: A mistake in a program.

Debugging: The process of eliminating bugs.

You will make many mistakes as you write programs. It's normal.



"As soon as we started programming, we found out to our surprise that it wasn't as easy to get programs right as we had thought. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."

– *Maurice Wilkes*

Impossible ideal: "Please compile, execute, and debug my program."

Why is this impossible? Stay tuned.

Bottom line: Programming is primarily a *process* of finding and fixing mistakes.

Debugging

is challenging because conditionals and loops *dramatically increase* the number of possible outcomes.

program structure	*no loops*	*n conditionals*	1 *loop*
number of possible execution sequences	1	2^n	no limit

Most programs contain *numerous* conditionals and loops, with nesting.

Good news. Conditionals and loops provide structure that helps us understand our programs.

Old and low-level languages have a *goto* statement that provides arbitrary structure. Eliminating *goto*s was controversial until Edsgar Dijkstra published the famous note "*Goto considered harmful* " in 1968.

→

" The quality of programmers is a decreasing function of the number of goto statements in the programs they produce. "

– *Edsgar Dijkstra*

Debugging your program: summary

Program development is a *four*-step process, with feedback.

EDIT your program.

COMPILE your program to create an executable file.

RUN your program to test that it works as you imagined.

TEST your program on realistic and real input data.

SUBMIT your program for independent testing and approval.

syntax error

runtime error

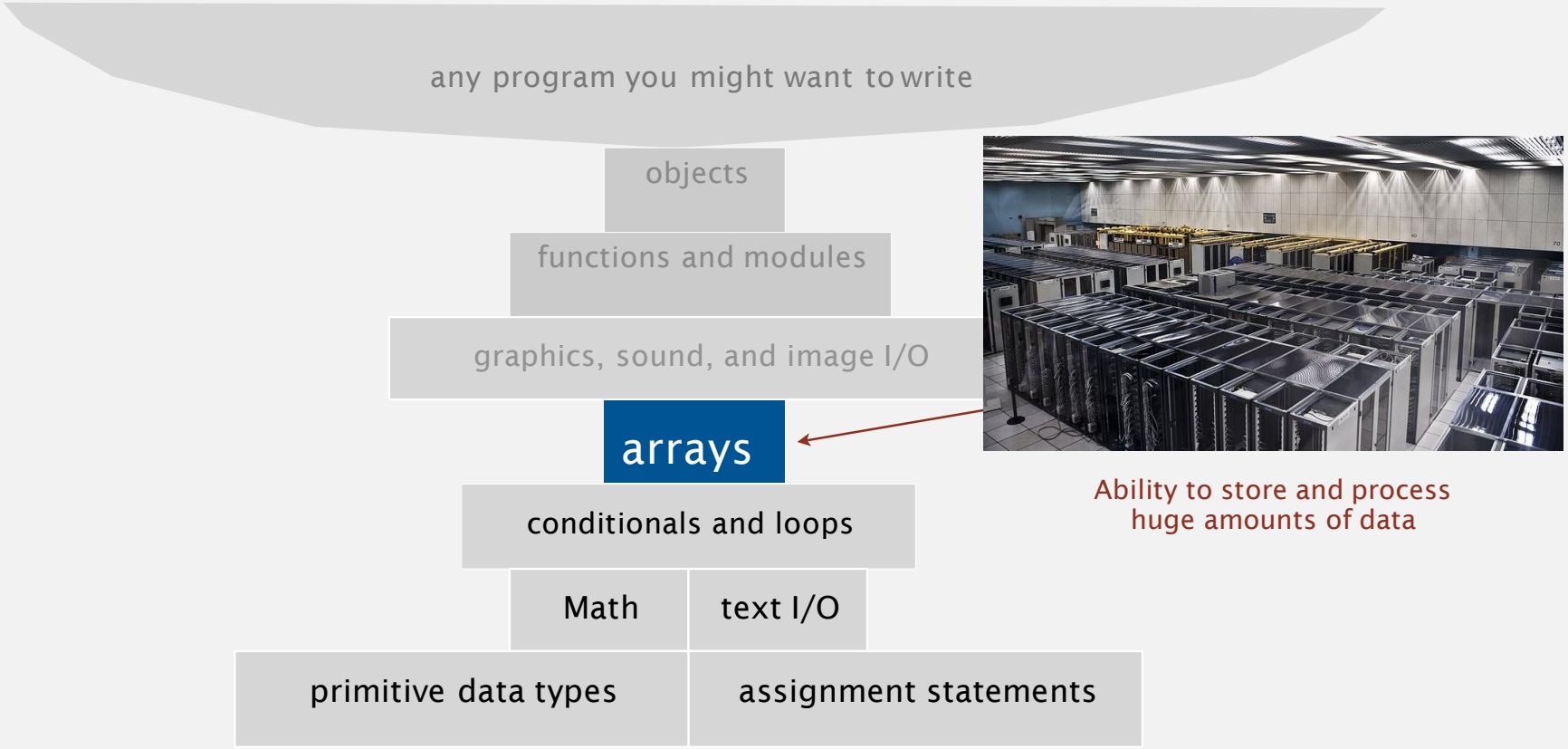
semantic error

performance error

Telling a computer what to do when you know what you're doing

Temel veri tipleri
Koşul ifadeleri ve döngüler
Diziler
Metotlar
Java ile Yaz, Derle, Çalıştır

Basic building blocks for programming



Ability to store and process huge amounts of data

Your first data structure

A data structure is an arrangement of data that enables efficient processing by a program.

An array is an *indexed* sequence of values of the same type.

Examples.

- 52 playing cards in a deck.
- 100 thousand students in an online class.
- 1 billion pixels in a digital image.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 86 billion neurons in the brain.
- 50 trillion cells in the human body.
- 6.02×10^{23} particles in a mole.

index	*value*
0	2♥
1	6♠
2	A♦
3	A♥
...	
49	3♣
50	K♣
51	4♠

Main purpose. Facilitate storage and manipulation of data.

Processing many values of the same type

10 values, without arrays

```
double a0 = 0.0;
double a1 = 0.0;
double a2 = 0.0;
double a3 = 0.0;
double a4 = 0.0;
double a5 = 0.0;
double a6 = 0.0;
double a7 = 0.0;
double a8 = 0.0;
double a9 = 0.0;
...
a4 = 3.0;
...
a8 = 8.0;
...
double x = a4 + a8;
```

tedious and error-prone code

10 values, with an array

```
double[] a;
a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
```

an easy alternative

1 million values, with an array

```
double[] a;
a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...
double x = a[234567] + a[876543];
```

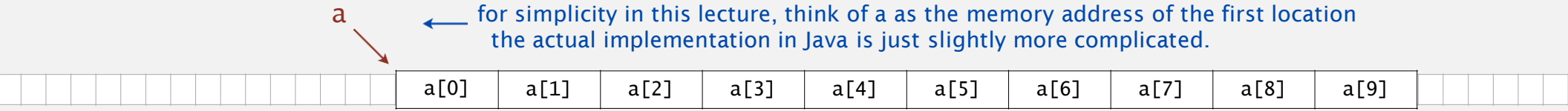
scales to handle huge amounts of data

Memory representation of an array

An array is an indexed sequence of values of the same type.

A computer's memory is *also* an indexed sequence of memory locations. ← stay tuned for many details

- Each primitive type value occupies a fixed number of locations.
- *Array values are stored in contiguous locations.*



Critical concepts

- Indices start at 0.
- Given `i`, the operation of accessing the value `a[i]` is extremely efficient.
- The assignment `b = a` makes the names `b` and `a` refer to the same array.

it does *not* copy the array, as with primitive types (stay tuned for details)

Java language support for arrays

Basic support

operation	*typical code*
Declare an array	`double[] a;`
Create an array of a given length	`a = new double[1000];`
Refer to an array entry by index	`a[i] = b[j] + c[k];`
Refer to the length of an array	`a.length;`

Initialization options

operation	*typical code*
Default initialization to 0 for numeric types	`a = new double[1000];`
Declare, create and initialize in one statement	`double[] a = new double[1000];`
Initialize to literal values	`double[] x = { 0.3, 0.6, 0.1 };`

no need to use a loop like

```
for (int i = 0; i < 1000; i++)
  a[i] = 0.0;
```

BUT cost of creating an array is proportional to its length.

Copying an array

To copy an array, create a new array , then copy all the values.

```
double[] b = new double[a.length];
for (int i = 0; i < a.length; i++)
   b[i] = a[i];
```

Important note: The code b = a does *not* copy an array (it makes b and a refer to the same array).

```
double[] b = new double[a.length];
b = a;
```

Programming with arrays: typical examples

Access command-line args in system array

```
int stake = Integer.parseInt(args[0]);
int goal = Integer.parseInt(args[1]);
int trials = Integer.parseInt(args[2]);
```

For brevity, N is `a.length` and `b.length` in all this code.

Copy to another array

```
double[] b = new double[N];
for (int i = 0; i < N; i++)
   b[i] = a[i];
```

Create an array with N random values

```
double[] a = new double[N];
for (int i = 0; i < N; i++)
   a[i] = Math.random();
```

Print array values, one per line

```
for (int i = 0; i < N; i++)
   System.out.println(a[i]);
```


Compute the average of array values

```
double sum = 0.0;
for (int i = 0; i < N; i++)
   sum += a[i];
double average = sum / N;
```

Find the maximum of array values

```
double max = a[0];
for (int i = 1; i < N; i++)
   if (a[i] > max) max = a[i];
```

Pop quiz 1 on arrays

Q. What does the following code print?

```
public class PQarray1
{
    public static void main(String[] args)
    {
        int[] a = new int[6];
        int[] b = new int[a.length];

        b = a;
        for (int i = 1; i < b.length; i++)
            b[i] = i;

        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();

        for (int i = 0; i < b.length; i++)
            System.out.print(b[i] + " ");
        System.out.println();
    }
}
```

Pop quiz 1 on arrays

Q. What does the following code print?

```
public class PQarray1
{
    public static void main(String[] args)
    {
        int[] a = new int[6];
        int[] b = new int[a.length];

        b = a;
        for (int i = 1; i < b.length; i++)
            b[i] = i;

        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();

        for (int i = 0; i < b.length; i++)
            System.out.print(b[i] + " ");
        System.out.println();
    }
}
```

After this, b and a refer to the same array

A.

```
% java PQ4_1
0 1 2 3 4 5

0 1 2 3 4 5
```

Programming with arrays: typical bugs

Array index out of bounds

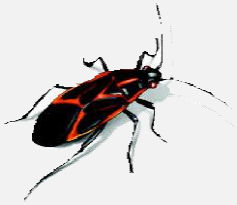
```
double[] a = new double[10];
for (int i = 1; i <= 10; i++)
   a[i] = Math.random();
```

No a[10] (and a[0] unused)

Uninitialized array

```
double[] a;
for (int i = 0; i < 9; i++)
   a[i] = Math.random();
```

Never created the array



Undeclared variable

```
a = new double[10];
for (int i = 0; i < 10; i++)
   a[i] = Math.random();
```

What type of data does a refer to?

Two-dimensional arrays

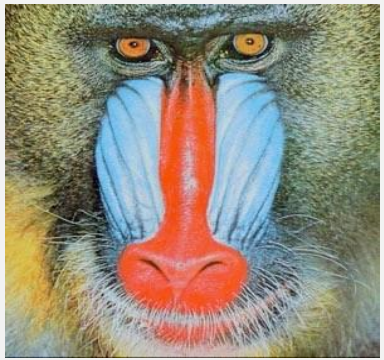
A two-dimensional array is a *doubly-indexed* sequence of values of the same type.

Examples

- Matrices in math calculations.
- Grades for students in an online class.
- Outcomes of scientific experiments.
- Transactions for bank customers.
- Pixels in a digital image.
- Geographic data
- ...

grade

student ID	0	1	2	3	4	5	...
0	A	A	C	B	A	C	
1	B	B	B	B	A	A	
2	C	D	D	B	C	A	
3	A	A	A	A	A	A	
4	C	C	B	C	B	B	
5	A	A	A	B	A	A	
...							



Main purpose. Facilitate storage and manipulation of data.

Java language support for two-dimensional arrays (basic support)

operation	*typical code*
Declare a two-dimensional array	`double[][] a;`
Create a two-dimensional array of a given length	`a = new double[1000][1000];`
Refer to an array entry by index	`a[i][j] = b[i][j] * c[j][k];`
Refer to the number of rows	`a.length;`
Refer to the number of columns	`a[i].length;` ← can be different for each row
Refer to row *i*	`a[i]` ← no way to refer to column j

a[][] →

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]	a[0][6]	a[0][7]	a[0][8]	a[0][9]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]	a[1][6]	a[1][7]	a[1][8]	a[1][9]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2][5]	a[2][6]	a[2][7]	a[2][8]	a[2][9]

a[1] →

a 3-by-10 array

Java language support for two-dimensional arrays (initialization)

operation	*typical code*
Default initialization to 0 for numeric types	`a = new double[1000][1000];`
Declare, create and initialize in a single statement	`double[][] a = new double[1000][1000];`
Initialize to literal values	`double[][] p = { { .92, .02, .02, .02, .02 }, { .02, .92, .32, .32, .32 }, { .02, .02, .02, .92, .02 }, { .92, .02, .02, .02, .02 }, { .47, .02, .47, .02, .02 }, };`

no need to use nested loops like

```
for (int i = 0; i < 1000; i++)
   for (int j = 0; j < 1000; j++)
      a[i][j] = 0.0;
```

BUT cost of creating an array is proportional to its size.

Application of arrays: vector and matrix calculations

Mathematical abstraction: vector

Java implementation: 1D array

Mathematical abstraction: matrix

Java implementation: 2D array

Vector addition

```
double[] c = new double[N];
for (int i = 0; i < N; i++)
   c[i] = a[i] + b[i];
```

Matrix addition

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      c[i][j] = a[i][j] + b[i][j];
```


.30	.60	.10	+	.50	.10	.40	=	.80	.70	.50

.70	.20	.10		.80	.30	.50		1.5	.50	.60
.30	.60	.10	+	.10	.40	.10	=	.40	1.0	.20
.50	.10	.40		.10	.30	.40		.60	.40	.80

Application of arrays: vector and matrix calculations

Mathematical abstraction: vector

Java implementation: 1D array

Mathematical abstraction: matrix

Java implementation: 2D array

Vector dot product

```
double sum = 0.0;
for (int i = 0; i < N; i++)
   sum += a[i]*b[i];
```

.30	.60	.10	·	.50	.10	.40	=	.25

i	x[i]	y[i]	x[i]*y[i]	sum
0	0.3	0.5	0.15	0.15
1	0.6	0.1	0.06	0.21
2	0.1	0.4	0.04	0.25

end-of-loop trace

Matrix multiplication

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

.70	.20	.10		.80	.30	.50		.59	.32	.41
.30	.60	.10	*	.10	.40	.10	=	.31	.36	.25
.50	.10	.40		.10	.30	.40		.45	.31	.42

Pop quiz 4 on arrays

Q. How many multiplications to multiply two *N*-by-*N* matrices?

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

1. N
2. N^2
3. N^3
4. N^4

Pop quiz 4 on arrays

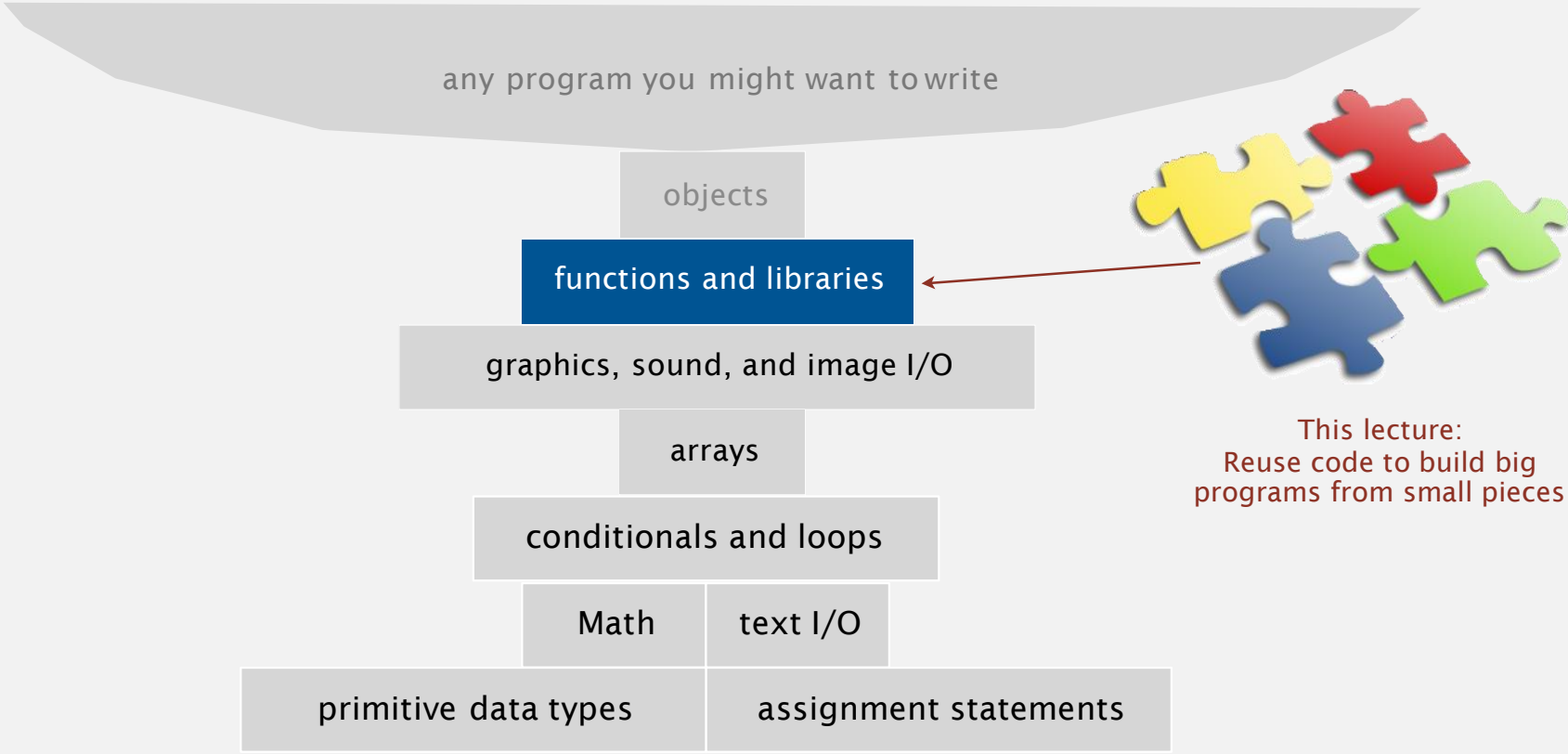
Q. How many multiplications to multiply two N-by-N matrices?

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

1. N
2. N^2
3. N^3 ← Nested for loops: $N \times N \times N$
4. N^4

Temel veri tipleri
Koşul ifadeleri ve döngüler
Diziler
Metotlar
Java ile Yaz, Derle, Çalıştır

Context: basic building blocks for programming



Functions, libraries, and modules

Modular programming

- Organize programs as independent modules that do a job together.
- Why? Easier to share and reuse code to build bigger programs.

Facts of life

- Support of modular programming has been a holy grail for decades.
- Ideas can conflict and get highly technical in the real world.

Def. A library is a set of functions.

↑ for purposes of this lecture

Def. A module is a `.java` file.

↑ for purposes of this course

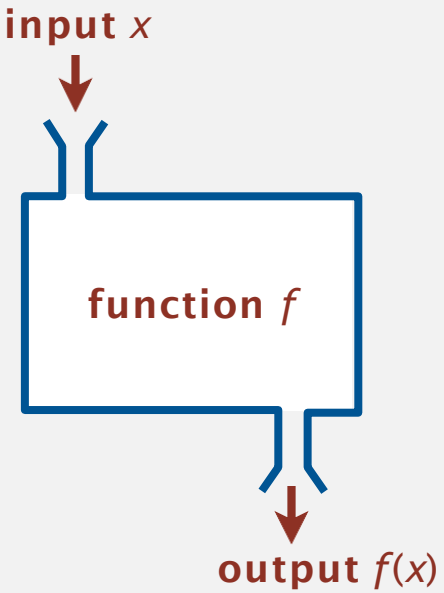
For now. Libraries and modules are the *same thing*: `.java` files containing sets of functions.

Later. Modules implement *data types* (stay tuned).

Functions (static methods)

Java *function* ("aka *static method*")

- Takes zero or more *input* arguments.
- Returns zero or one *output* value.



Applications

- Scientists use mathematical functions to calculate formulas.
- Programmers use functions to build modular programs.
- *You* use functions for both.

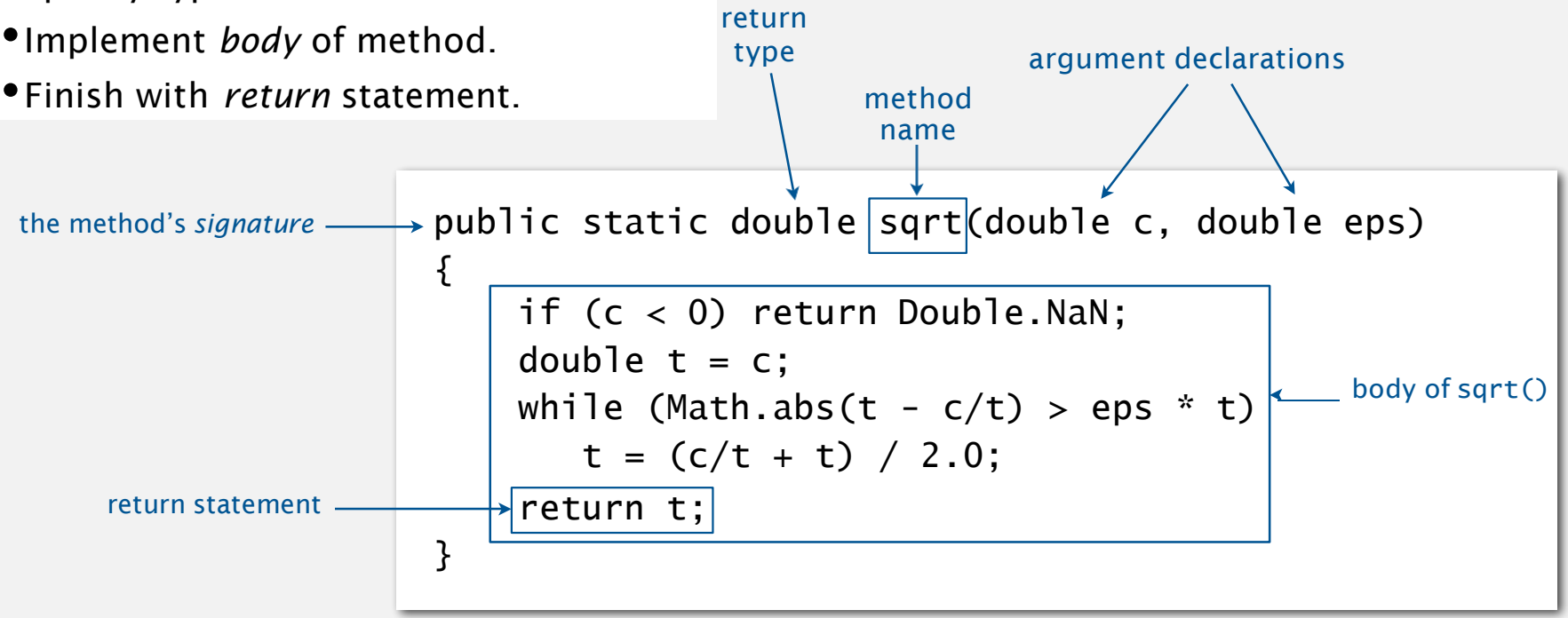
Examples seen so far

- Built-in functions: `Math.random()`, `Math.abs()`, `Integer.parseInt()`.
- Our I/O libraries: `StdIn.readInt()`, `StdDraw.line()`, `StdAudio.play()`.
- User-defined functions: `main()`.

Anatomy of a Java static method

To implement a function (static method)

- Create a *name*.
- Declare type and name of *argument*(*s*).
- Specify type for *return value*.
- Implement *body* of method.
- Finish with *return* statement.



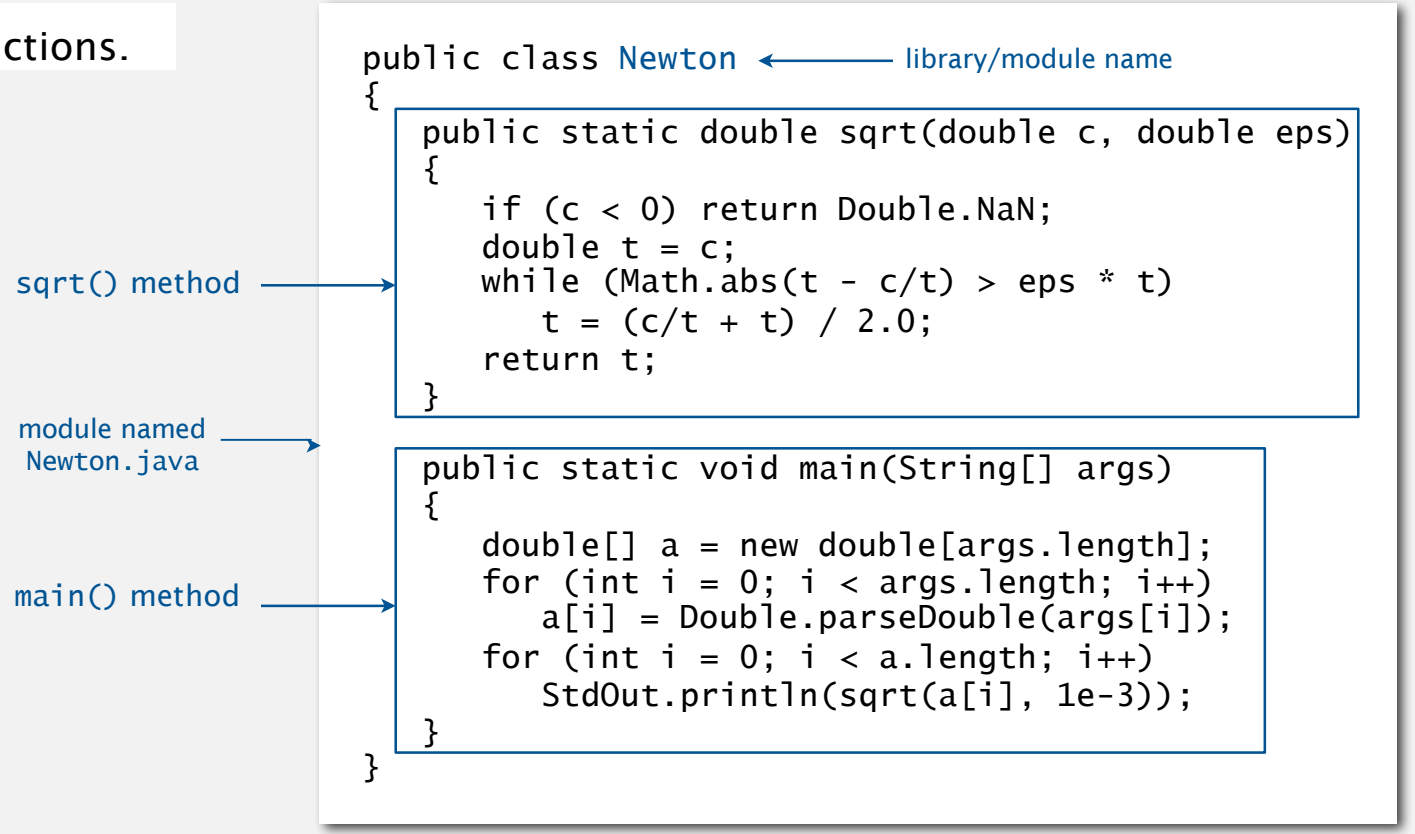
Anatomy of a Java library

A library is a set of functions.

Note: We are using our `sqrt()` from Lecture 2 here to illustrate the basics with a familiar function.

Our focus is on control flow here. See Lecture 2 for technical details.

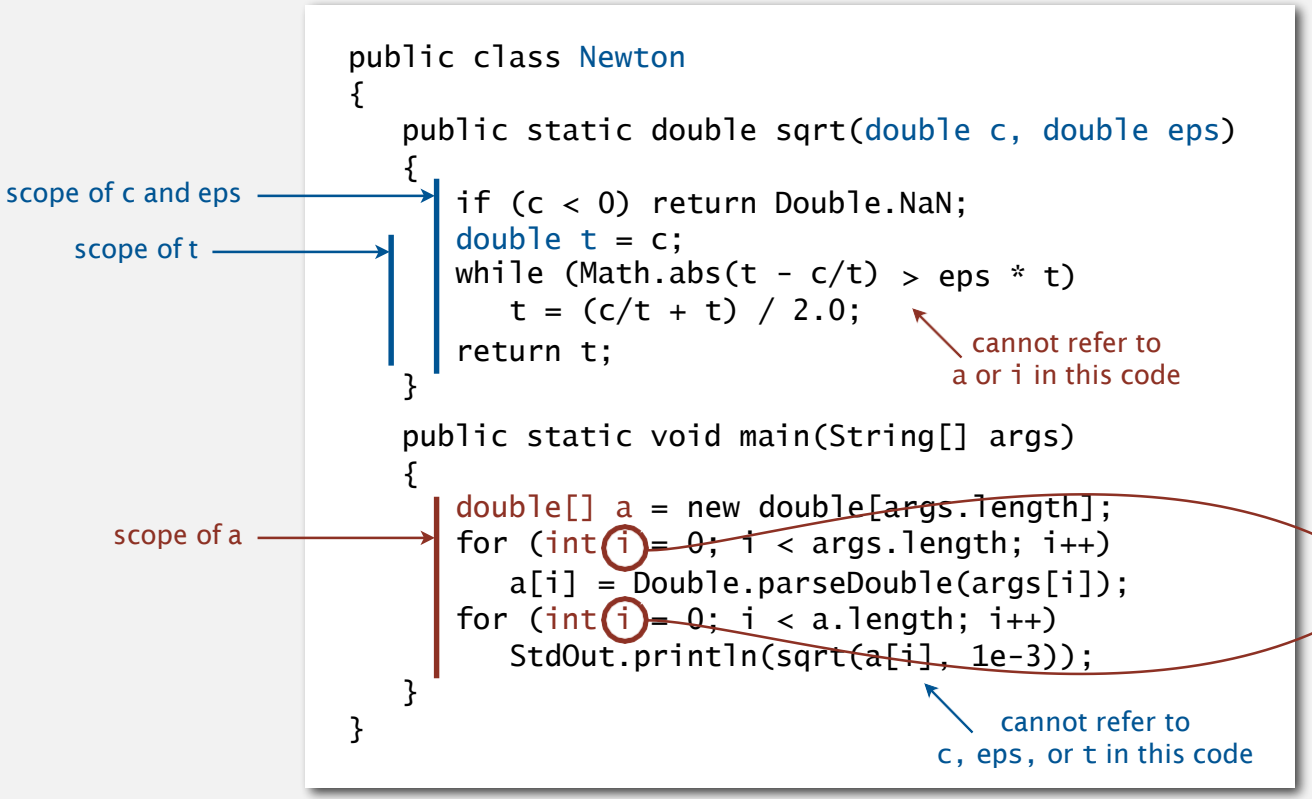
You can use `Math.sqrt()`.



Key point. Functions provide a *new way* to control the flow of execution.

Scope

Def. The scope of a variable is the code that can refer to it by name.



In a Java library, a variable's scope is the code following its declaration, in the same block.

Best practice. Declare variables so as to *limit* their scope.

Flow of control

```
public class Newton
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * t)
         t = (c/t + t) / 2.0;
      return t;
   }
   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
      {
         double x = sqrt(a[i], 1e-3);
         StdOut.println(x);
      }
   }
}
```

Summary of flow control for a function call

- Control transfers to the function code.
- Argument variables are declared and initialized with the given values.
- Function code is executed.
- Control transfers back to the calling code (with return value assigned in place of the function name in the calling code).

"pass by value"
(other methods used in other systems)

Note. OS calls `main()` on `java` command

Function call flow of control trace

```
public class Newton
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * t)
         t = (c/t + t) / 2.0;
      return t;
   }
   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
      {
         double x = sqrt(a[i], 1e-3);
         StdOut.println(x);
      }
   }
}
```

c	t
3.0	3.0
	2.0
	1.75
	1.732

i	a[i]	x
0	1.0	1.000
1	2.0	1.414
2	3.0	1.732
3		

```
% java Newton 1 2 3
1.000
1.414
1.732
```

Pop quiz 1a on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1a
{
   public static int cube(int i)
   {
      int j = i * i * i;
      return j;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1a on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1a
{
   public static int cube(int i)
   {
      int j = i * i * i;
      return j;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Takes *N* from the command line, then prints cubes of integers from 1 to *N*

```
% javac PQfunctions1a.java
% java PQfunctions1a 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Pop quiz 1b on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1b
{
   public static int cube(int i)
   {
      int i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1b on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1b
{
   public static int cube( int i )
   {
      int i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Won't compile. Argument variable i is declared and initialized for function block, so the name cannot be reused.

```
% javac PQfunctions1b.java
PQfunctions1b.java:5: i is already defined in cube(int)
      int i = i * i * i;
          ^
1 error
```

Pop quiz 1c on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1c
{
   public static int cube(int i)
   {
      i = i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1c on functions

Q. What happens when you compile and run the following code?

```
public class PQ6_1c
{
   public static int cube(int i)
   {
      i = i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Won't compile. Need `return` statement.

```
% javac PQfunctions1c.java
PQfunctions1c.java:6: missing return statement
   }
   ^
1 error
```

Pop quiz 1d on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1d
{
   public static int cube(int i)
   {
      i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1d on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1d
{
   public static int cube(int i)
   {
      i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Works. The `i` in `cube()` is

- Declared and initialized as an argument.
- Different from the `i` in `main()`.

BUT changing values of function arguments is sufficiently confusing to be deemed bad style for this course.

```
% javac PQfunctions1d.java
% java PQfunctions1d 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Pop quiz 1e on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1e
{
   public static int cube(int i)
   {
      return i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

Pop quiz 1e on functions

Q. What happens when you compile and run the following code?

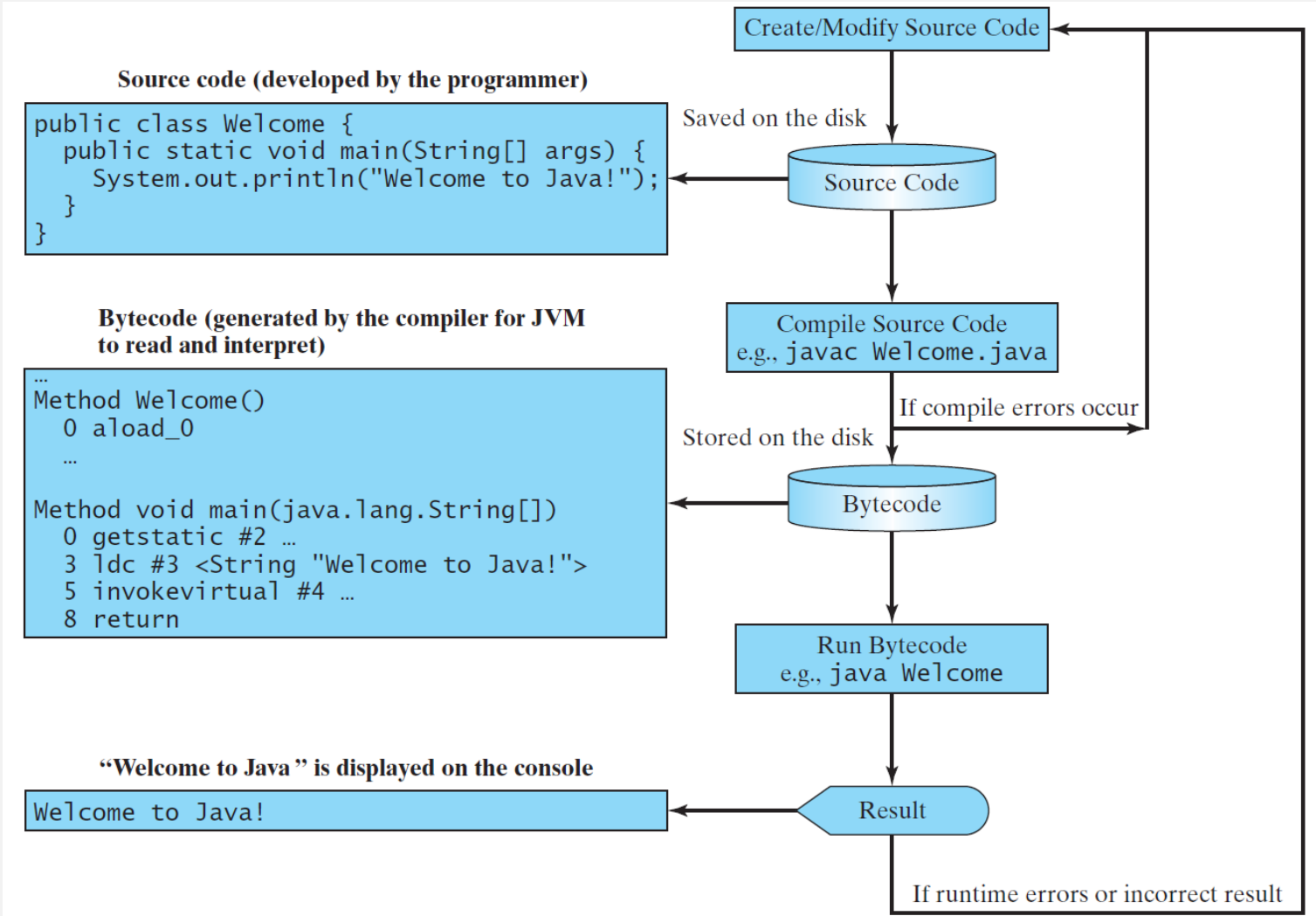
```
public class PQfunctions1e
{
   public static int cube(int i)
   {
      return i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

A. Works fine. Preferred (compact) code.

```
% javac PQfunctions1e.java
% java PQfunctions1e 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Temel veri tipleri
Koşul ifadeleri ve döngüler
Diziler
Metotlar
Java ile Yaz, Derle, Çalıştır

Java ile Yaz, Derle, Çalıştır



Java ile Yaz, Derle, Çalıştır