

Lecture 1-1:

A Taste of Computer Science

About this lecture

- In order to get a feeling for CS, I will show some sophisticated stuff
- But ... there is no need to understand the details of anything in this lecture!
- Objective: to give some taste of the spirit of CS

Dedicated to Prof. Barbara Liskov



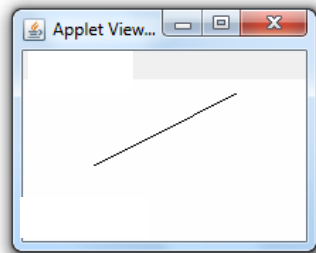
Winner of the 2009 Turing Award

Drawing a line in Java

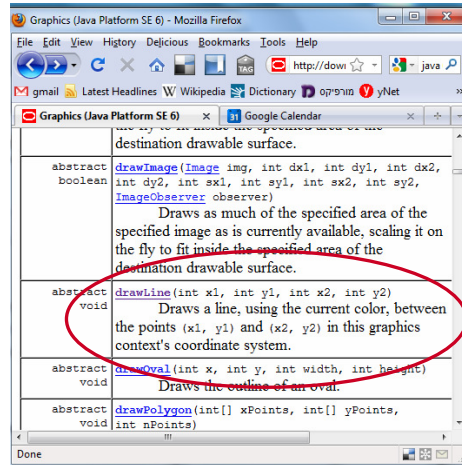
Java program:

```
// Some Java code ...  
page.drawLine (50, 60, 150, 10);  
// java code continues ...
```

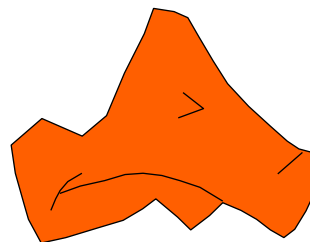
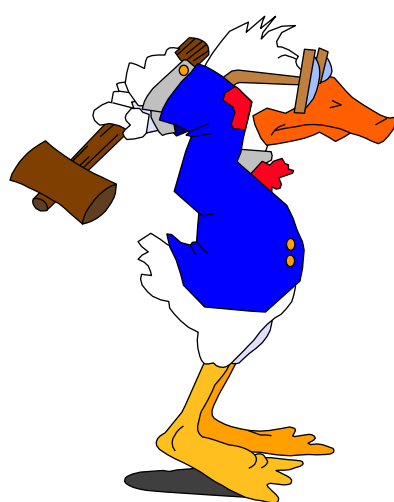
Result:



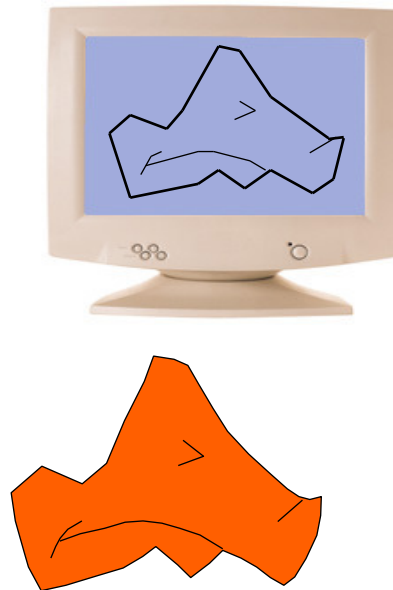
Java documentation (excerpt)



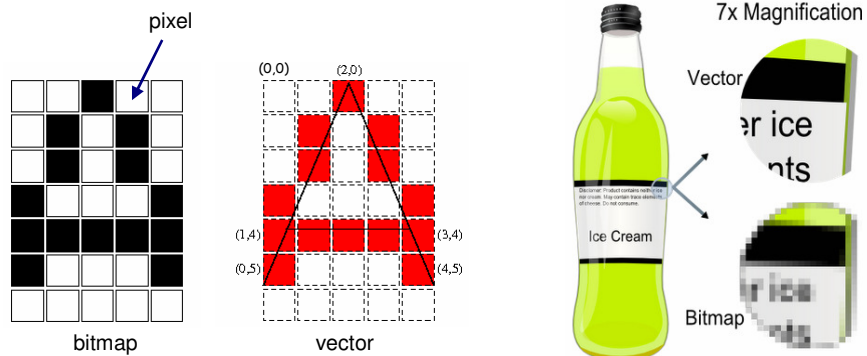
Drawing a figure in PowerPoint



Drawing a figure in PowerPoint



Computer graphics: bitmap versus vector

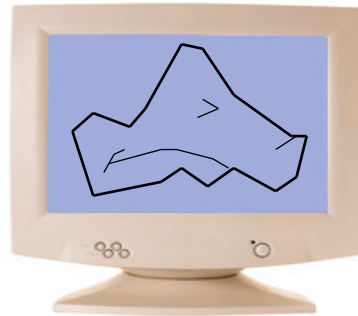


- Bitmap file: 00100 01010 01010 10001 11111 10001 00000, ...
- Vector graphics file: drawLine(2,0,0,5), drawLine(2,0,4,5), drawLine(1,4,3,4)
- In both methods, the only available primitive operation is **drawPixel(x,y)**
- Each method has its pros and cons
- In this lecture we will focus on **vector graphics**.

It's all about line drawing

- How to draw a line?
- How to draw a line *fast*?

Remember: the only available operation is `drawPixel(x,y)`



Simplify the problem

Let's focus only on lines that go north-east.

It's all about line drawing

- How to draw a line?
- How to draw a line *fast*?

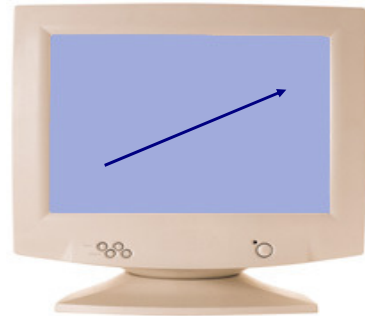
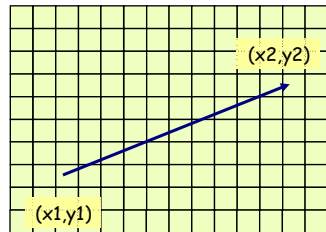
Remember: the only available operation is `drawPixel(x,y)`



Simplify the problem

Let's focus only on lines that go north-east.

Primitive operation: `drawPixel(x,y)`



(0,0)

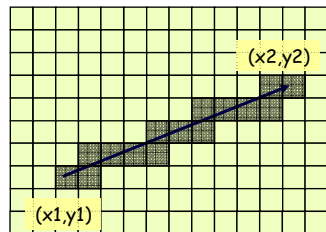
(Using a coordinate system with a conventional origin, to make things more intuitive)

Simplify the problem

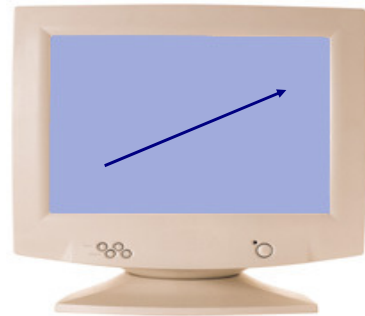
Let's focus only on lines that go north-east.

Task: implement `drawLine(x1,y1,x2,y2)` where $x_2 > x_1$ and $y_2 > y_1$, when the only available operation is `drawPixel(x,y)`

Primitive operation: `drawPixel(x,y)`



(0,0)



Simplify the problem

Let's focus only on lines that go north-east.

Task: implement `drawLine(x1,y1,x2,y2)` where $x_2 > x_1$ and $y_2 > y_1$, when the only available operation is `drawPixel(x,y)`

Insight: Because we always go north-east, in each step we either go north (up), or east (right).

`drawLine(x1,y1,x2,y2)`

(0,0)

➔

`drawLine(x,y,x+dx,y+dy)`

$dx = x_2 - x_1$
 $dy = y_2 - y_1$

Use a good problem description

This can make a huge difference.

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 11

Let variables **a** and **b** record how many pixels you went right and up, respectively

<code>a=0, b=0</code>	<code>drawPixel(x+0,y+0)</code>
<code>a=1, b=0</code>	<code>drawPixel(x+1,y+0)</code>
<code>a=1, b=1</code>	<code>drawPixel(x+1,y+1)</code>
<code>a=2, b=1</code>	<code>drawPixel(x+2,y+1)</code>
<code>a=3, b=1</code>	<code>drawPixel(x+3,y+1) ...</code>

In general:
`a=a+1 or b=b+1 drawPixel(x+a,y+b)`

`drawLine(x,y,x+dx,y+dy)`

$dx = x_2 - x_1$
 $dy = y_2 - y_1$

set `a=0, b=0`

While there is more work to do:

`drawPixel(x+a,y+b)`

// set up for drawing the next pixel:

to go right, set `a=a+1`

to go up, set `b=b+1`

loop

Use a formal language

That's where programming enters the picture.

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 12

Let variables **a** and **b** record how many pixels you went right and up, respectively

```

a=0, b=0    drawPixel(x+0,y+0)
a=1, b=0    drawPixel(x+1,y+0)
a=1, b=1    drawPixel(x+1,y+1)
a=2, b=1    drawPixel(x+2,y+1)
a=3, b=1    drawPixel(x+3,y+1) ...

In general:
a=a+1 or b=b+1  drawPixel(x+a,y+b)

```

`drawLine(x,y,x+dx,y+dy)`

↓

```

set a=0, b=0
While there is more work to do: ?
  drawPixel(x+a,y+b)
  // set up for drawing the next pixel:
  to go right,  set a=a+1
  to go up,    set b=b+1

```

→

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  to go right:  a=a+1
  to go up:    b=b+1

```

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 13

`drawLine(x,y,x+dx,y+dy)`

Think abstractly

Look at the slopes.

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  to go right:  a=a+1
  to go up:    b=b+1

```

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 14

(x, y) $(x+a, y+b)$ $(x+dx, y+dy)$

dx dy

$\frac{dy}{dx}$

`drawLine(x,y,x+dx,y+dy)`

x, y $(x+dx, y+dy)$

$dx = x_2 - x_1$ $dy = y_2 - y_1$

Think abstractly

Look at the slopes.

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  to go right:  a=a+1
  to go up:    b=b+1

```

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 15

(x, y) $(x+a, y+b)$ $(x+dx, y+dy)$

dx dy

$\frac{dy}{dx}$

a b

Think abstractly

Look at the slopes.

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  to go right:  a=a+1
  to go up:    b=b+1

```

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 16

$b/a > dy/dx$:
overshooting

(x, y) $(x+a, y+b)$ $(x+dx, y+dy)$

dx dy

$\frac{b}{a}$ $\frac{dy}{dx}$

Think abstractly

Look at the slopes.

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a, y+b)
  // set up for the next pixel:
  to go right: a=a+1
  to go up:   b=b+1
  
```

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com slide 17

$b/a > dy/dx$:
overshooting

(x, y) $(x+a, y+b)$ $(x+dx, y+dy)$

dx dy

$\frac{b}{a}$ $\frac{dy}{dx}$

$a = a + 1$

Think abstractly

Look at the slopes.

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a, y+b)
  // set up for the next pixel:
  to go right: a=a+1
  to go up:   b=b+1
  
```

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com slide 18

$b/a > dy/dx$:
overshooting

(x, y) $(x+dx, y+dy)$

dx dy

$a = a + 1$

b/a dy/dx

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  if b/a > dy/dx then a=a+1
                      else b=b+1

```

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  to go right: a=a+1
  to go up:   b=b+1

```

Think abstractly
Look at the slopes.

Analyze:
The algorithm's run-time

Optimize:
Minimize run-time

$b/a > dy/dx$ implies $b \cdot dx > a \cdot dy$
Let: $diff = b \cdot dx - a \cdot dy$

Observations about diff:

$a = b = 0$ implies $diff = 0$

$b/a > dy/dx$ implies $diff > 0$

Therefore, instead of saying
if $b/a > dy/dx$ we can say if $diff > 0$

When we set $a = a + 1$,
we get $diff = diff - dy$

When we set $b = b + 1$,
we get $diff = diff + dx$

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  if b/a > dy/dx then a=a+1
                      else b=b+1

```

Division! Oy Vey!

Perhaps
we can get rid
of the division
operations

Analyze:

The algorithm's run-time

Optimize:

Minimize run-time

$b/a > dy/dx$ implies $b*dx > a*dy$

Let: $diff = b*dx - a*dy$

Observations about diff:

$a = b = 0$ implies $diff = 0$

$b/a > dy/dx$ implies $diff > 0$

Therefore, instead of saying
if $b/a > dy/dx$ we can say if $diff > 0$

When we set $a = a + 1$,
we get $diff = diff - dy$

When we set $b = b + 1$,
we get $diff = diff + dx$

```
a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  if b/a > dy/dx then a=a+1
  else b=b+1
```



```
a=0, b=0
diff = b*dx - a*dy
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  if diff > 0 then a=a+1, diff = diff - dy
  else b=b+1, diff = diff + dx
```

Additions only!

How good is the algorithm?

Run-time analysis

- To draw a line consisting of n pixels, the algorithm performs n addition operations
- The fastest line drawing algorithm possible!

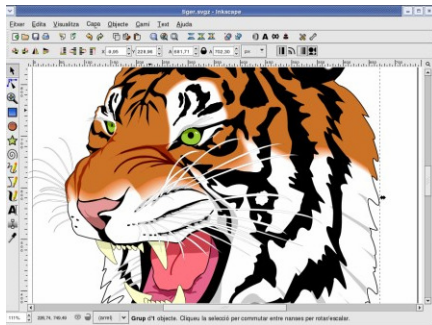


```
a=0, b=0
diff = b*dx - a*dy
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  if diff > 0 then a=a+1, diff = diff - dy
  else b=b+1, diff = diff + dx
```

Implications of the algorithm

Bottom up

- Now we can draw lines fast; therefore,
- We can draw polygons fast; therefore,
- We can draw images fast; therefore,
- We can draw 30 frames per second; therefore,
- We have digital graphics, animation, video, . . .

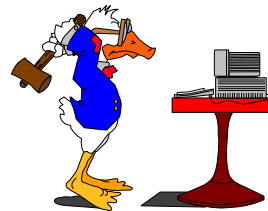


Top down

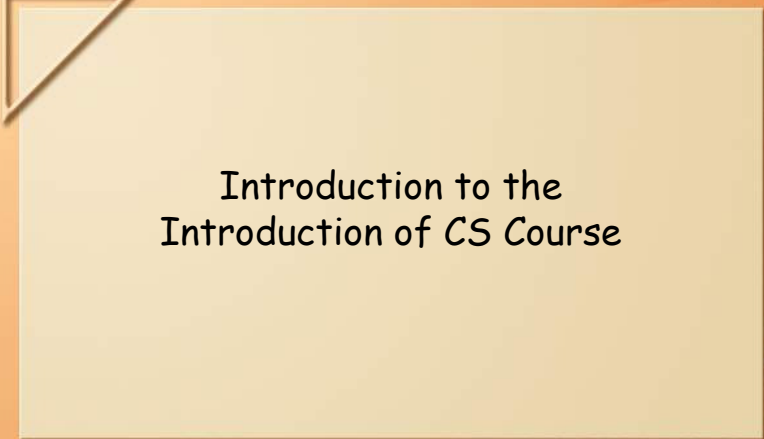
- Much of vector graphics is drawing polygons
- To draw polygons, you need to draw lines
- To draw lines, you need to divide
- Division can be re-expressed as multiplication
- Multiplication can be reduced to addition
- Addition is fast.

Lessons for the beginning computer scientist

- Abstraction: when faced with a complex problem, ignore the details and focus on solving the essence
- Simplicity: Divide the problem into manageable sub-problems
- Reduction: Try to re-express the problem in terms of another problem that you know how to handle
- Expression: Invent and use a clean and sharp notation
- Efficiency: In many cases, the solution must be as efficient as possible
- Elegance: Whatever you do, try to keep it beautiful



Much of computer science is about abstraction, simplicity, reduction, expression, efficiency, and elegance.



Introduction to the Introduction of CS Course

Course objectives

Upon completion of this course you will become ...

- An informed Computer Science student
- A competent Java programmer
- Ready for subsequent CS courses.

In addition, you will ...

- Sharpen your analytic skills
- Begin to learn to think and plan clearly and elegantly
- Develop a taste for beauty in science and engineering
- Learn how to learn and develop.

Course rules

Requirements

- Attend / follow two weekly lectures + one weekly recitation
- Read the assigned pages in the text book
- Submit a weekly homework assignment
- Visit the course web site at least once a day
- **Important:** the only formal and abiding channel of communications between the course team and the students is the course web site.

How to ask questions and get help (after class)

- Post your question on the course web site - you'll get an answer within 24 hours
- The answer will be posted by the course team, or by another student
- Think twice before you post anything
- Visit the lab in the days/times listed in the course web site; A tutor from the course team will be there to help.

Collaboration / cheating policy

- All homework assignments must be completed individually
- It is encouraged to talk to other students about the homework assignments
- But, this exchange should be restricted to talking only
- Once you start writing something together on a piece of paper or on the computer, you are crossing the line from collaboration to cheating
- **A very easy rule to remember:** any exchange of written notes about HW between students (hand-written, email, whatever) is cheating
- Cheaters will be prosecuted.

Course resources

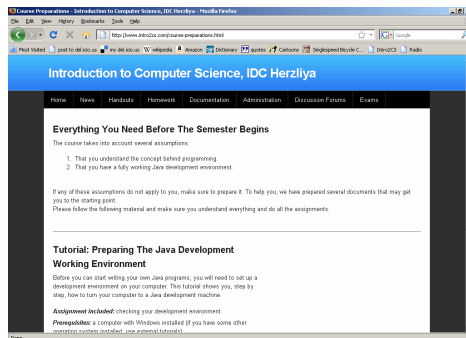
Course team:

Instructor: Shimon Schocken

Teaching Assistant: Boaz Kantor

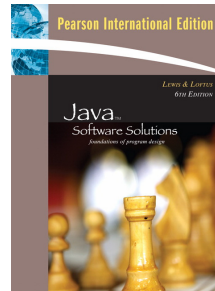
Chief Tutor: Steven Karass

Course web site:



www.intro2cs.com

Course textbook:




Java Software Solutions: Foundations of Program Design, by Lewis and Loftus, 6th edition (earlier editions are fine also), Publisher: Pearson / Addison Wesley

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com

slide 29

The Java Tutorials – do it yourself!

**The Java™ Tutorials**

Search
Feedback

The Java Tutorials are practical guides for programmers who want to use the Java programming language to create applications. They include hundreds of complete, working examples, and dozens of lessons. Groups of related lessons are organized into "trails".

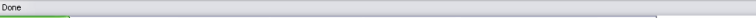
Trails Covering the Basics


These trails are available in book form as *The Java Tutorial, Fourth Edition*. To buy this book, refer to the box to the right.

- ▶ **Getting Started** — An introduction to Java technology and lessons on installing Java development software and using it to create a simple program.
- ▶ **Learning the Java Language** — Lessons describing the essential concepts and features of the Java Programming Language.
- ▶ **Essential Java Classes** — Lessons on exceptions, basic input/output, concurrency, regular expressions, and the platform environment.
- ▶ **Collections** — Lessons on using and extending the Java Collections Framework.
- ▶ **Swing** — An introduction to the Swing GUI toolkit, with an overview of features and a visual catalog of components. See below for a more comprehensive tutorial on Swing.
- ▶ **Deployment** — How to package applications and applets using JAR files, and deploy them using Java Web Start and Java Plug-in.
- ▶ **Preparation for Java Programming Language Certification** — List of available training and tutorial resources.

Creating Graphical User Interfaces

Done



[License Info](#)

Software

- ▶ [The Java Development Kit \(JDK\)](#) for Java SE 6.
- ▶ [NetBeans IDE](#)
- ▶ [Java EE SDK](#)

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com

slide 30