Coding

Bertram Capital Management
Professional Development
March 2, 2021

Agenda

Part 1:

- Why code?
- What is code?
- Example

Part 2:

- Building blocks
- Solutions
- Example

Coding

Part 1

Who should learn to code?

Part 1

EVERYONE

Part 1

Knowing how to code….

- **Demystifies** everyday technology and makes the modern world more **understandable**
- Provides a tool for **makers**, **problem solvers**, and anyone with a “**big idea**”
- Gives you access to **advanced features** of **tools you already use**
- Gives you **career flexibility**
- Reinforces **logical thinking** and **keeps you sharp**
- Is **empowering** and builds **confidence**

Coding is awesome because….

- There are **logical outcomes**
- You get to **solve puzzles**
- You can create **something** from **nothing**
- You can work **individually** or as part of a **team**
- You can have ***TOTAL CONTROL***
- There can be (nearly) **instant gratification**
- Your work can be used/experienced **by other people**
- You have can distribute your work **world-wide**
- **IT'S SUPER FUN**

What is coding?

- Getting a **computer/device** to **do what you want** it to do
- Writing instructions about **what to do** and **when**
- Writing instructions about **how to get from start to finish**
- Transforming **inputs into outputs**
  - *Inputs*: on/off switch, time, data, sensors, user activity
  - *Outputs*: on/off switch, data, graphics, streams, almost anything!
- **Translating human instructions into instructions a computer/device can understand**
  - Best/Worst Kid Ever: Does exactly what you say

Code is written as text, usually in a plain ol' text file

Part 1

Examples of coding

- Emojis

  `:)` -> 😀

- Skype bold command

  `*Hello*` -> **Hello**

- Excel formulas

  `=SUM(A2:A8)`-> 37

- HTML markup

  `<i>Hello</i>` -> *Hello*

- database queries

  `SELECT F_NAME || L_NAME FROM PEOPLE` -> Jerry Seinfeld

- CloudFormation AWS specs

```
MyDataBucket:
 Type: AWS::S3::Bucket
 Properties:
   AccessControl: AuthenticatedRead
   BucketName: my-data-bucket
```

- NodeJS AWS Lambda microservice

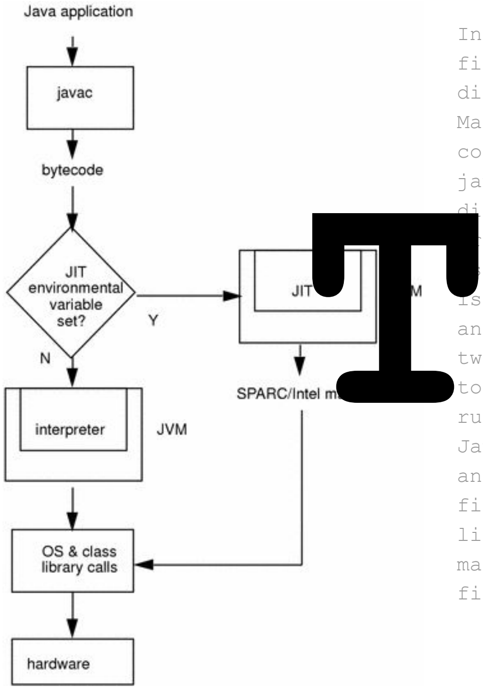
```
module.exports.list = (evt, ctx, cc) => {
  console.log('list',event);
  let output = {status: 200, data: [],
    meta: {}, message:null};
    return esClient.search({
    …
    });
};
```

Context is everything

Where will your code runs will determine...

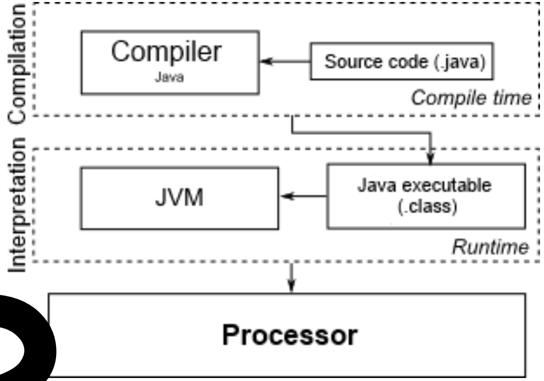
- Languages/instruction syntax
- Possible commands/capabilities
- Execution rules
- Available helpers (libraries)
- Execution performance
- Access to computer resources
- Installation/distribution

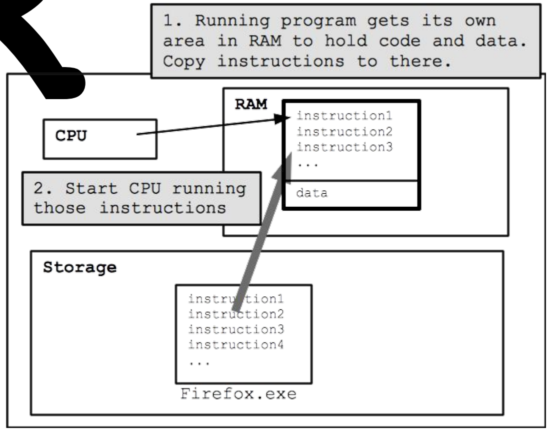
How does my code run?



In Java, programs are not compiled into executable files; they are compiled into bytecode (as discussed earlier), which the JVM (Java Virtual Machine) then executes at runtime. Java source code is compiled into bytecode when we use the javac compiler. The bytecode gets saved on the disk with the file extension .class. When the program is to be run, the bytecode is converted, using the just-in-time (JIT) compiler. The result is machine code which is then fed to the memory and is executed. Java code needs to be compiled twice in order to be executed. Java programs need to b[illegible] when [illegible]ecode is run, it needs to be converted to machine code. The Java classes/bytecode are compiled to machine code and loaded into memory by the JVM when needed the first time. This is different from other languages like C/C++ where programs are to be compiled to machine code and linked to create an executable file before it can be executed.

TLDNR




1. Running program gets its own area in RAM to hold code and data. Copy instructions to there.

2. Start CPU running those instructions

Part 1

Example

Stand-alone Guestbook

Part 1

Coding

Part 2

Building Blocks

- Data Stores
  Relational databases (Oracle, MS-SQL)
  Object/No-SQL databases (Mongo, DynamoDB)
  Search indexes (ElasticSearch, Solr)
  Files (XML, JSON, MS Excel, CSV, etc)
- Asynchronous Activities
  Work queues (RabbitMQ, Amazon SQS)
  Broadcast topics (RabbitMQ, Amazon SNS)
  Timers (cron, Amazon CloudWatch events)
- Web Servers
  Traditional (Apache HTTP, Nginx, Tomcat)
  CDN (CloudFlare, CloudFront)
  Embedded (Oracle, ElasticSearch)
- Business Logic Executors
  Dedicated Servers (WebSphere, WebLogic)
  Microservices (NodeJS, AWS Lambda, Google Cloud Functions)
  Embedded (Web browsers, Web servers, Datastores, Messaging systems)
- User Interfaces
  Desktop/mobile web browser (Chrome, Safari)
  Desktop/mobile app (MS Excel, Instagram)
  Other devices (Apple Watch)
- Out-of-app Communication
  External APIs
  Email
  SMS
- Other Equipment
  Sensors (Temperature, acceleration, location)
  Switches (Motors, power outlets, locks)

Solutions

- Decompose problems into aspects/concerns
  What is the "end game" functionality we need?
  What is the "implied" functionality we need?
  What models/analogs/solutions already exist?
  What are the current/future standards/best-practices we need to follow?
- Map aspects/concerns to building blocks
  What components that do what we need to do?
  Can we think about functionality differently to create "better" mappings?
  What components must be built and what can be bought and configured?
- Compose selected building blocks into a solution
  Will we have everything we need?
  Will all the components work together?
  What will the total cost of ownership be?
  Can the solution evolve over time?
- Break-down the work that needs to be done
  Who will do the work?
  How long will it take?
  How much will it cost?

Getting work done (bleh)

Development Process

- Analysis
- Planning
- Design
- Construction
- Testing
- Deploying

Additional Activities

- Status monitoring
- Time tracking
- Invoicing
- Customer feedback

Part 2

Example

Web Guestbook

Part 2