COMP 251

University of the Fraser Valley

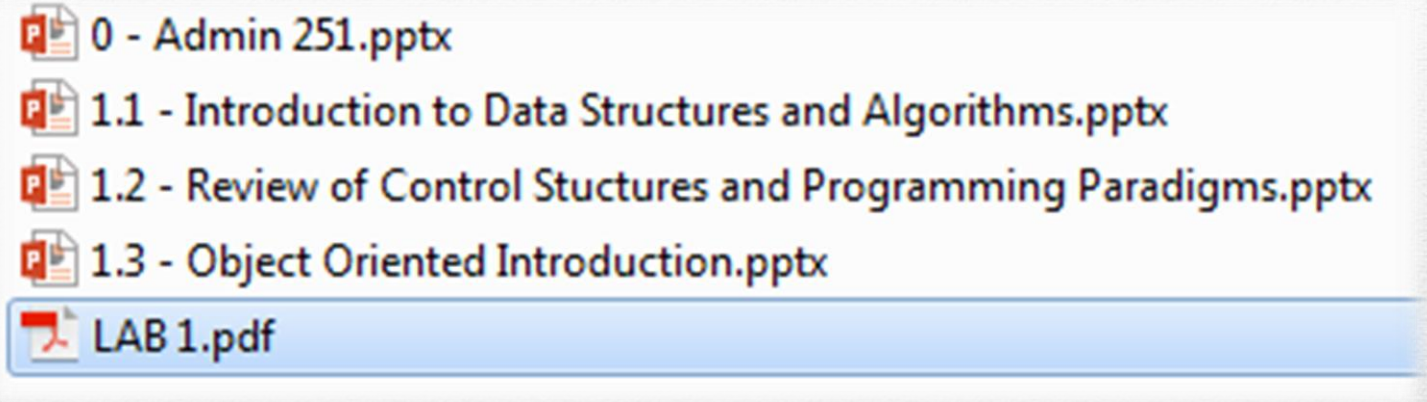
DATA TYPES AND OPERATORS

1. What are the three fundamental *control structures* of computer programs?
2. What is a data structure?
3. What are the main characteristics of OOP?
4. What is an intractable function?
5. **Compare** and **contrast** *procedural and OO* programming with one **word each**.

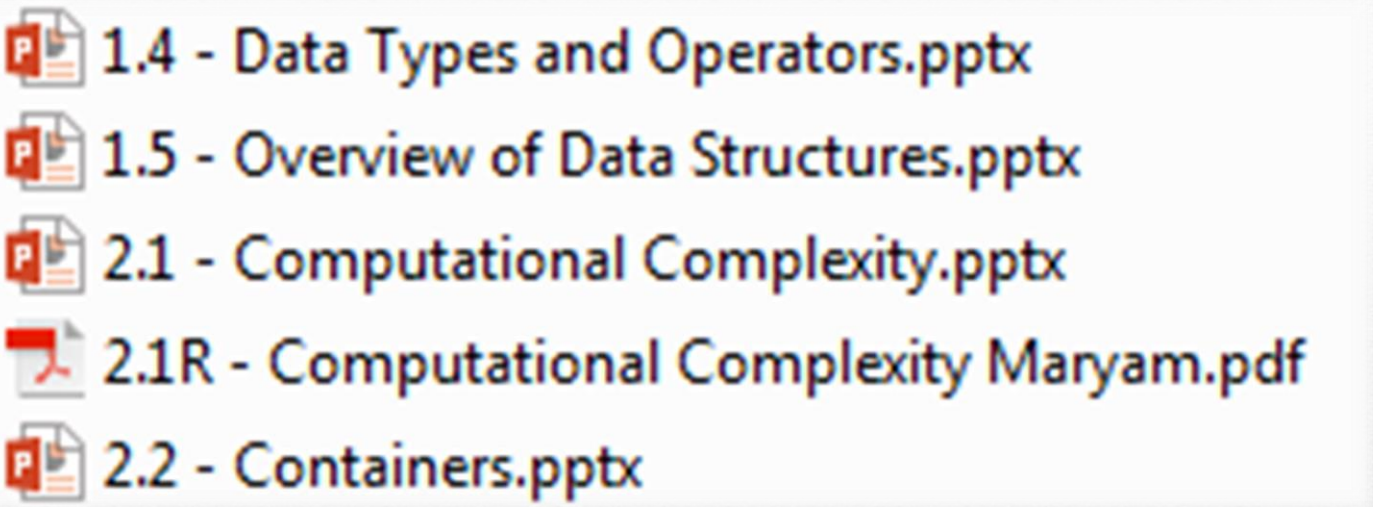
Quiz

Tentative Course Outline

Week/ Date	Topic	Chapters	Assignment Due
Week 1 May 1st	Introduction and Review of Prerequisite Material	First four chapters (Part 1)	Lab 1
Week 2 May 8th and 10th	Algorithms and Building Blocks (Algorithmic Analysis and Application Programming Interface in the context of Java" - "Collections API")	5 and 6	Homework 1 No Lab
Week 3 May 15th and 17th	Recursion Implementing *ArrayList* and Linked Lists	7, 15 and 17	Lab 2
Week 4 May 22nd and 24th	**MIDTERM** (22nd based on first 3 weeks) Stacks and Queues, Application of Stacks	16 and 11	Homework 2 Lab 2 Continued
Week 5 May 29th and 31st	Introduction to Trees, Binary Trees, Binary Search Trees, AVL Trees.	18 and 19	Lab 3
Week 6 June 5th and 7th	Binary Heap, Priority Queues and Sorting Algorithms	21 and 8	Homework 3 Lab 3 Continued
Week 7 June 12th and 14th	Hashing and Graphs	20 and 14	Lab 4
Week 8 Monday June 20th	Review		


0 - Admin 251.pptx
1.1 - Introduction to Data Structures and Algorithms.pptx
1.2 - Review of Control Stuctures and Programming Paradigms.pptx
1.3 - Object Oriented Introduction.pptx
LAB 1.pdf

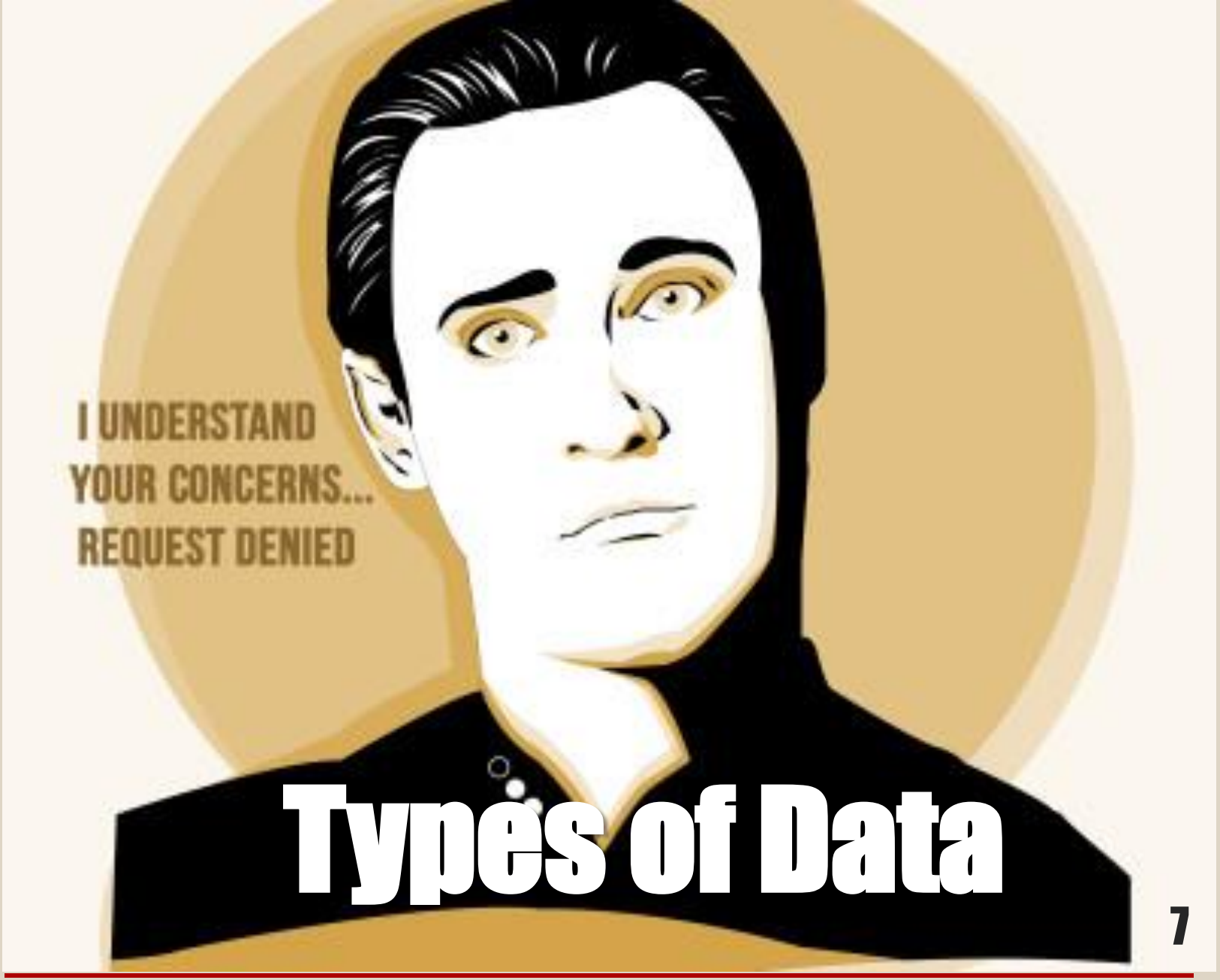
Last Week



Objectives for Week 2

Objectives for This PPT

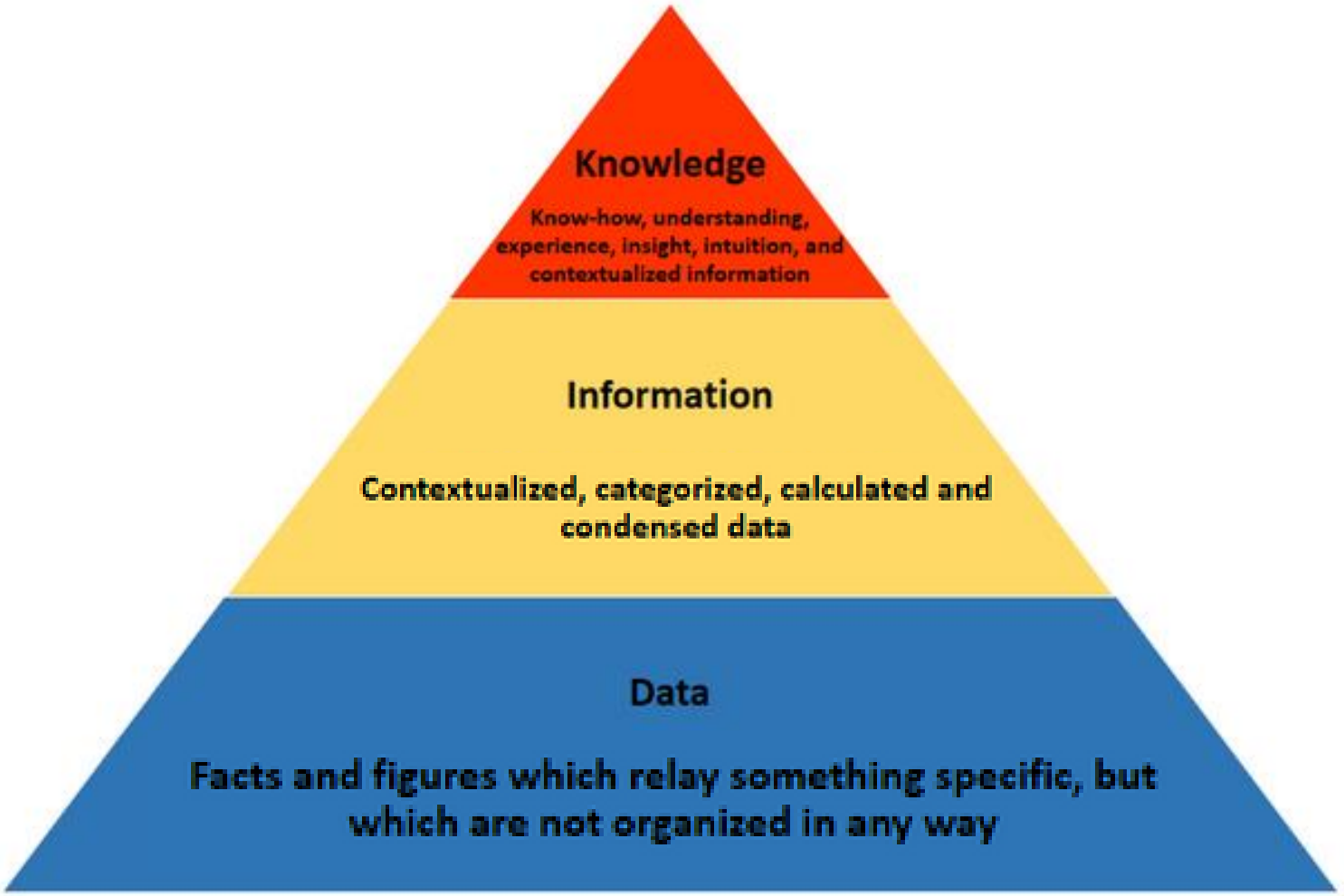
- Introduce the **Theory of Data** in Computing Science.
- Define **data types** and **data structures**, and contrast them.
- Discuss constants, variables, operators and expressions.
- Define arithmetic/logic operators in Java explicitly.
- Discuss **data typing** and **dynamic typing**.
- Discuss "**upcasting**" and "**downcasting**" in Java.
- Discuss **non-numerical data** (characters and Boolean variables) in Java.
- Define **Abstract Data Types (ADTs)**.



Types of Data

First,

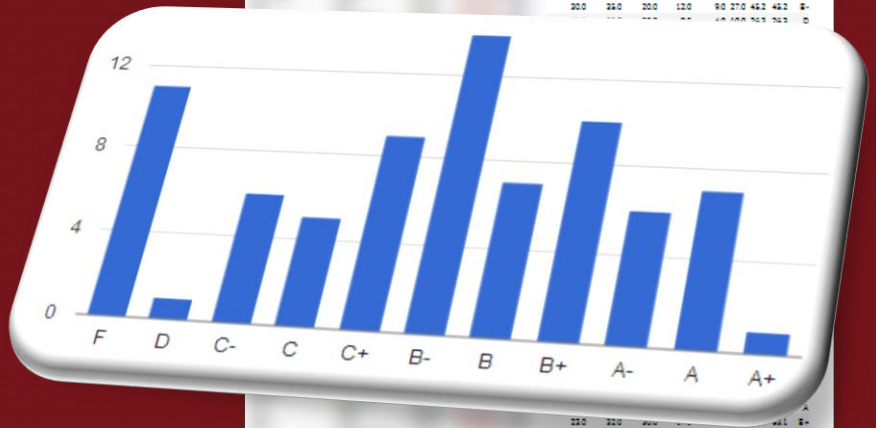
What is Data?



Taxonomy

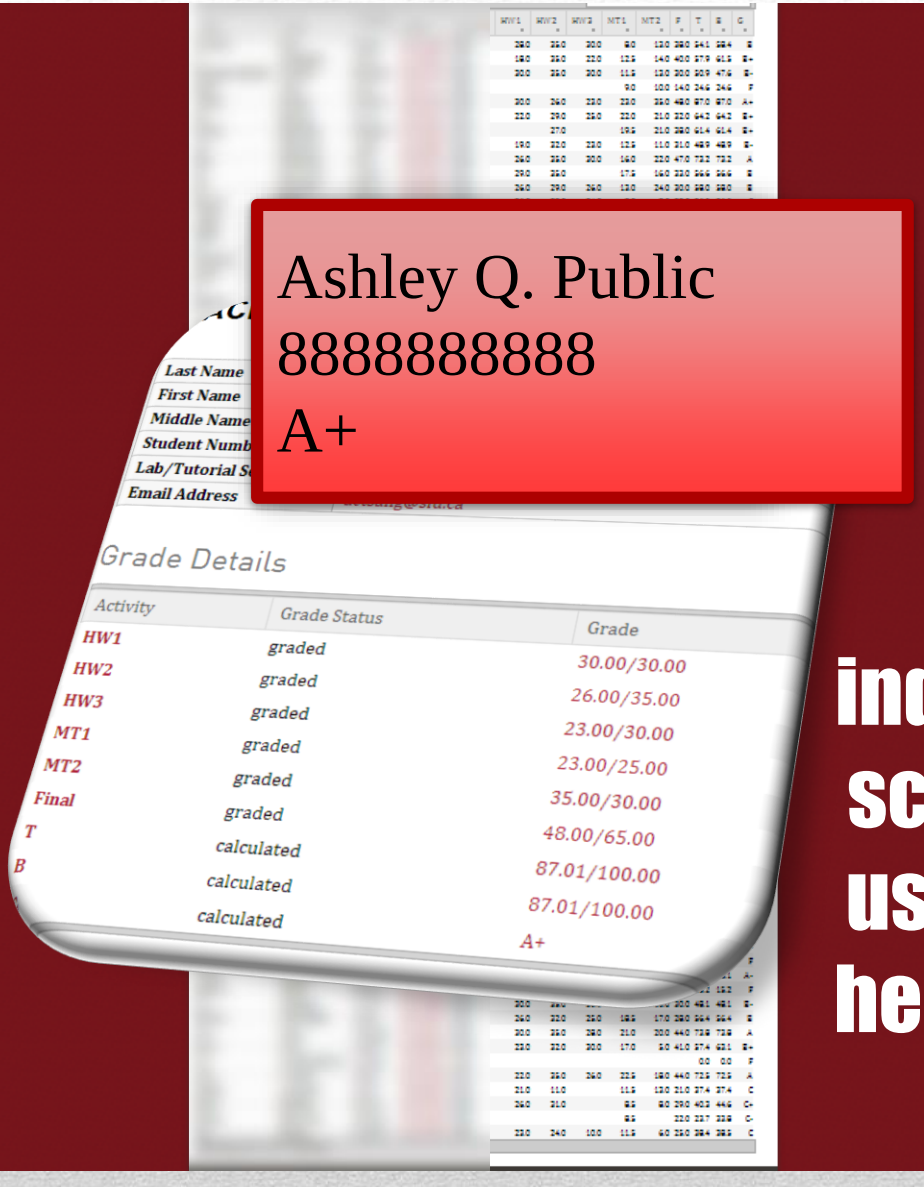
http://www.knowledge-management-tools.net/knowledge-information-data.html

Ashley Q. Public
8888888888
A+



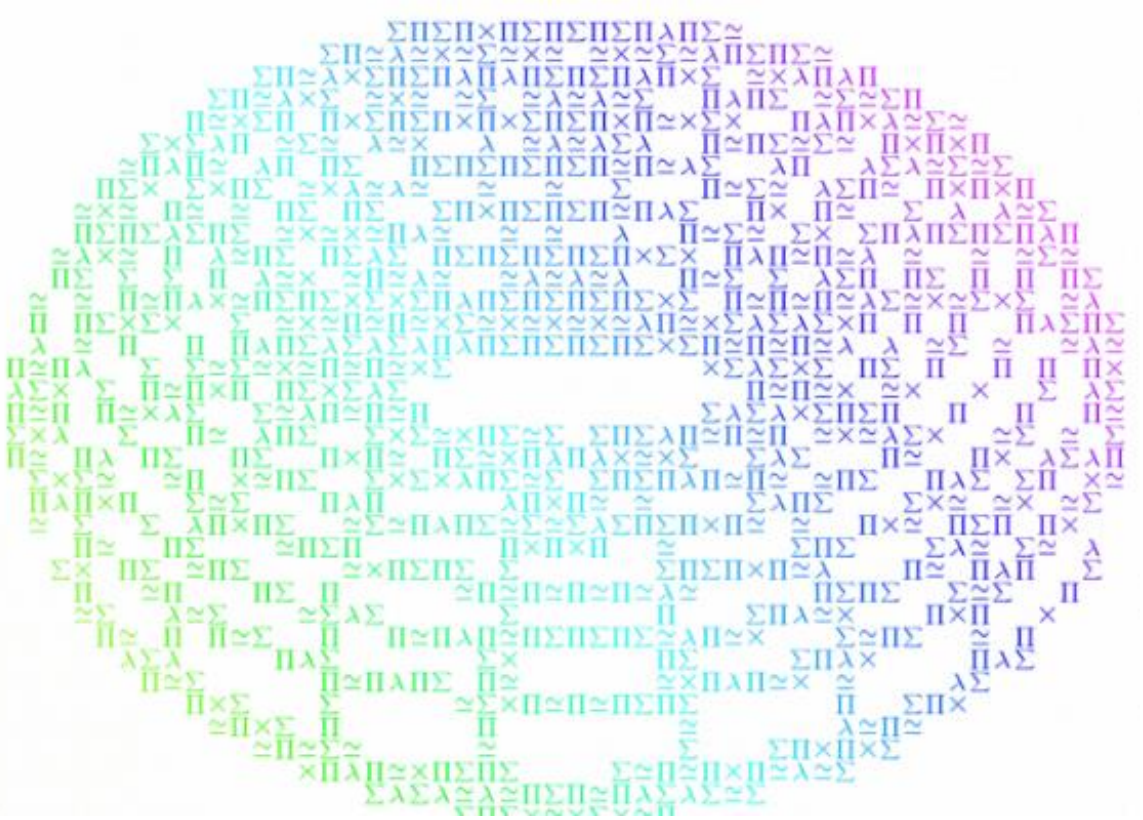
Example:

Ashley's grade is a datum used to calculate COMP 251 statistics



...whereas her individual testing scores were data used to calculate her course grade.

Theory of Data



Type Theory and Type Systems

Type Theory:

- In mathematics, logic, and computer science, a **type theory** is any of a class of formal systems, some of which can serve as alternatives to set theory as a foundation for all mathematics. In type theory, every "term" has a "type" and operations are restricted to the terms of a certain type.
- Type theory is closely related to (and in some cases overlaps with) type systems, which are a programming language feature used to reduce bugs. The types of type theory were created to avoid paradoxes **(*i.e.*, Russell's Paradox)** in a variety of formal logics and *rewrite systems* and sometimes "type theory" is used to refer to this broader application.

Definition

Type System:

- A set of rules that assigns a property (called a *type*) to the elements of a computer program (*i.e.*, variables, expressions, functions, *etc.*) for the purpose of reducing coding errors.

Definition

Type System:

- A set of rules that assigns a property (called a *type*) to the elements of a computer program (*i.e.*, variables, expressions, functions, *etc.*) for the purpose of reducing coding errors.
- In OO, **typing** is the enforcement of a class/type of an object to ensure that the object is manipulated by valid expressions (expressions are formally defined later in this PPT).

Definition

Type System:

- A set of rules that assigns a property (called a *type*) to the elements of a computer program (*i.e.*, variables, expressions, functions, *etc.*) for the purpose of reducing coding errors.
- In OO, **typing** is the enforcement of a class/type of an object to ensure that the object is manipulated by valid expressions (expressions are formally defined later in this PPT).
- Depending on the extent of enforcement and the precise instant of enforcement, there are **strong/weak** and **static/dynamic** typings respectively – to be defined and discussed later in this PPT.

Definition


INTEGER
NUMERIC
TEXT
DATE
ARRAY
DECIMAL
CHAR
BOOLEAN
TIME
SERIAL
UUID
INTERVAL
VARCHAR
TIMESTAMP

Data Types

Almost all programming languages explicitly include the notion of data type, though different languages may use different terminology. Common data types include:

- integers
- booleans
- characters
- floating-point numbers
- alphanumeric strings

Definition: Data Type

Almost all programming languages explicitly include the notion of data type, though different languages may use different terminology. Common data types include:

- integers
- booleans
- characters
- floating-point numbers
- alphanumeric strings

For example, in the Java programming language, the "int" type represents the set of 32-bit integers ranging in value from -2,147,483,648 to 2,147,483,647, as well as the operations that can be performed on integers, such as addition, subtraction, and multiplication. Colors, on the other hand, are represented by three bytes denoting the amounts each of red, green, and blue, and one string representing that color's name; allowable operations include addition and subtraction, but not multiplication.

Definition: Data Type

Further Definitions

Data Types come in the following categories:

- **Primitive**, which can be either:
  - A *basic type* is a data type provided by a programming language as a basic building block. Most languages allow more complicated *composite types* to be recursively constructed starting from basic types.
  - A *built-in type* is a data type for which the programming language provides built-in support.
- **Non-Primitive** which can be either:
  - Composite structures of primitive types such as an array.
  - Data types that are not defined by the programming language, but are instead created by the programmer.

From the **Java** perspective:

In Java, there is a very clear distinction between primitive and non-primitive types.

A variable of a primitive type directly contains the value of that type (in other words, they are *value types*).

A variable of a non-primitive type doesn't contain the value directly; instead, it is a *reference* (similar to a pointer) to an object. (It is not possible in Java to create user-defined value types).

Java has eight primitive types: `byte`, `short`, `int`, `long`, `char`, `boolean`, `float` and `double`. Anything else is a non-primitive type.

Java Specific

- The simple examples of data types given three slides back are, in fact, the common *primitive data types* utilized in computer programs include the following:
  1. Integers,
  2. Booleans,
  3. Characters,
  4. Floating Point Numbers,
  5. Alphanumeric Strings.

Data Types

Data Types

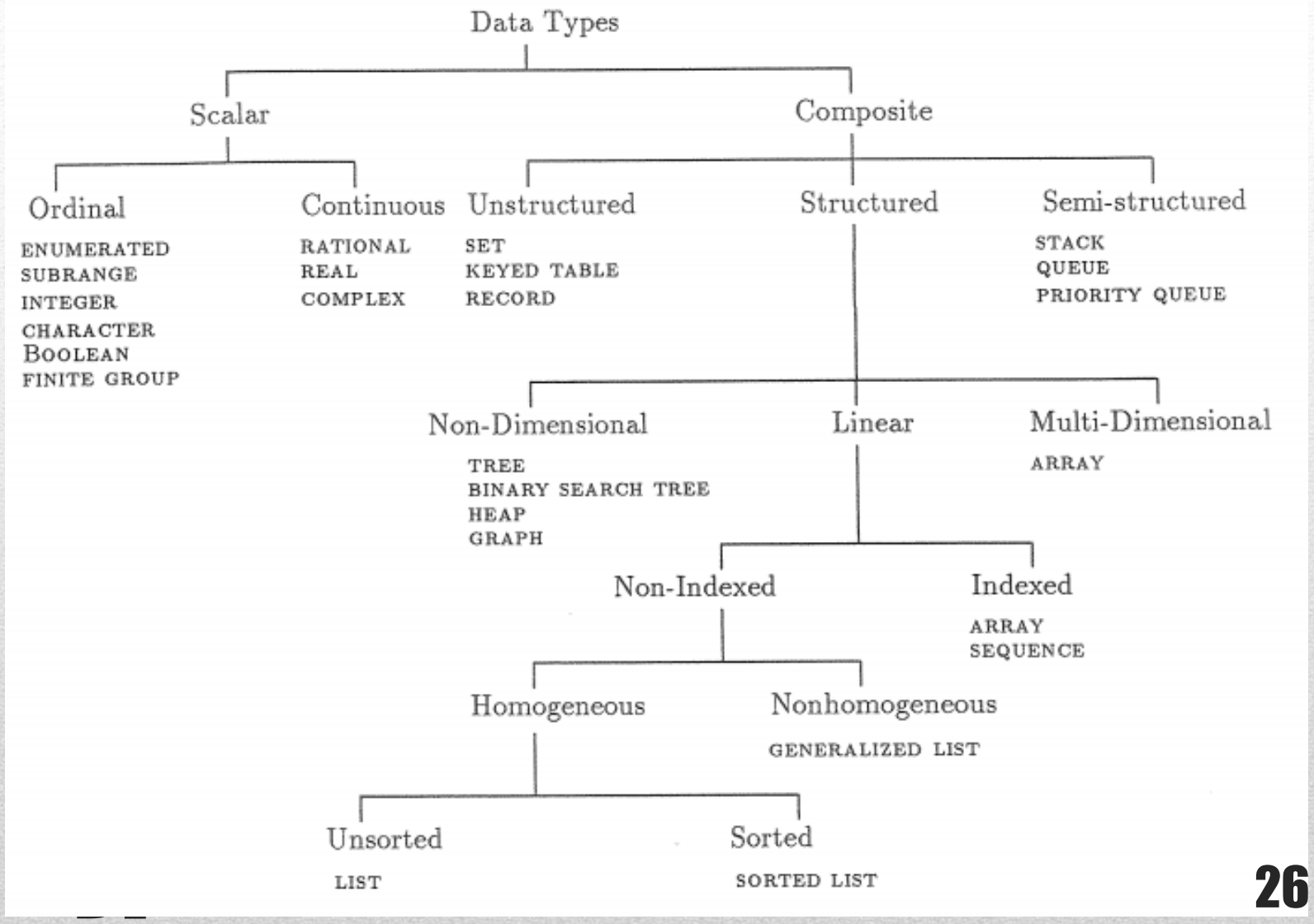
- The simple examples of data types given three slides back are, in fact, the common *primitive data types* utilized in computer programs include the following:
  1. Integers,
  2. Booleans,
  3. Characters,
  4. Floating Point Numbers,
  5. Alphanumeric Strings..
- One can go further and produce *complex/composite data types* (*i.e.*, an array), which are built upon simpler data types (*i.e.*, array elements), that are related to *data structures*.

Primitive data types in Java

The eight primitive data types in Java are:

- boolean, the type whose values are either true or false
- char, the character type whose values are 16-bit Unicode characters
- The arithmetic types:
  - The integral types:
    - byte
    - short
    - int
    - long
  - The floating-point types:
    - float
    - double

http://cs.fit.edu/~ryan/java/language/java-data.html


Data Types
Scalar
Composite
Ordinal
ENUMERATED
SUBRANGE
INTEGER
CHARACTER
BOOLEAN
FINITE GROUP
Continuous
RATIONAL
REAL
COMPLEX
Unstructured
SET
KEYED TABLE
RECORD
Structured
Semi-structured
STACK
QUEUE
PRIORITY QUEUE
Non-Dimensional
TREE
BINARY SEARCH TREE
HEAP
GRAPH
Linear
Multi-Dimensional
ARRAY
Non-Indexed
Indexed
ARRAY
SEQUENCE
Homogeneous
Nonhomogeneous
GENERALIZED LIST
Unsorted
LIST
Sorted
SORTED LIST

stackoverflow

Scalar vs. primitive data type - are they the same thing?

71

In various articles I have read, there are sometimes references to primitive data types and sometimes there are references to scalars.

My understanding of each is that they are data types of something simple like an int, boolean, char, etc.

Note: Scalar Data Types

stackoverflow

Scalar vs. primitive data type - are they the same thing?

71

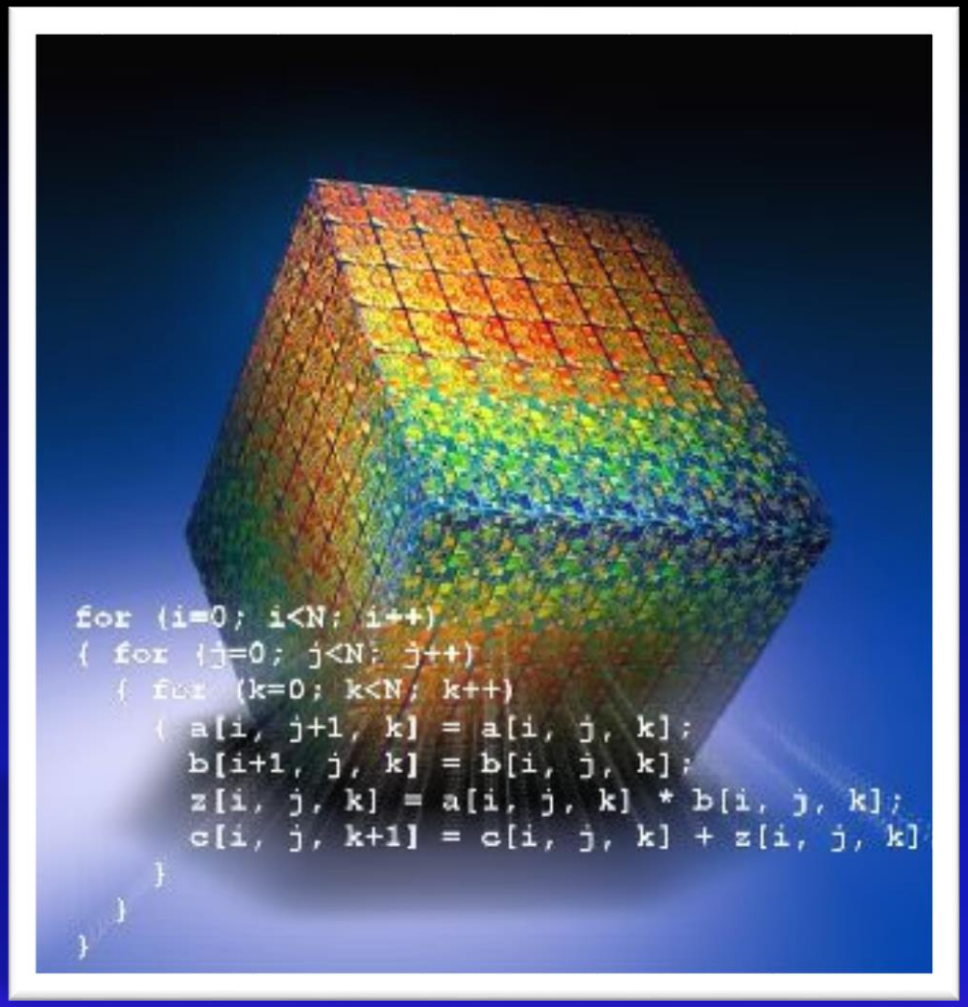
In various articles I have read, there are sometimes references to primitive data types and sometimes there are references to scalars.

My understanding of each is that they are data types of something simple like an int, boolean, char, etc.

Scalars are typically contrasted with *compounds*, such as arrays, maps, sets, structs, etc. A scalar is a "single" value - integer, boolean, perhaps a string - while a compound is made up of multiple scalars (and possibly references to other compounds). "Scalar" is used in contexts where the relevant distinction is between single/simple/atomic values and compound values.

Primitive types, however, are contrasted with e.g. *reference types*, and are used when the relevant distinction is "Is this directly a value, or is it a reference to something that contains the real value?", as in Java's primitive types vs. references. I see this as a somewhat lower-level distinction than scalar/compound, but not quite.

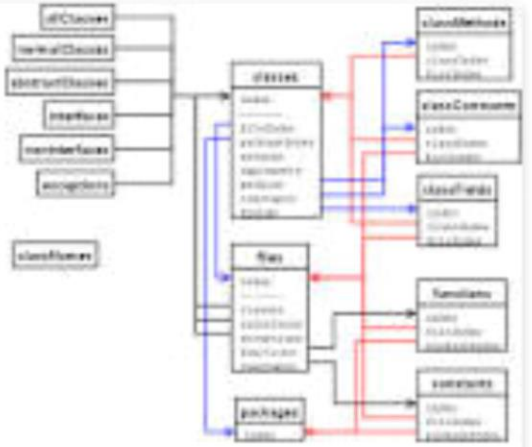
Note: Scalar Data Types



Aside on Data Structures

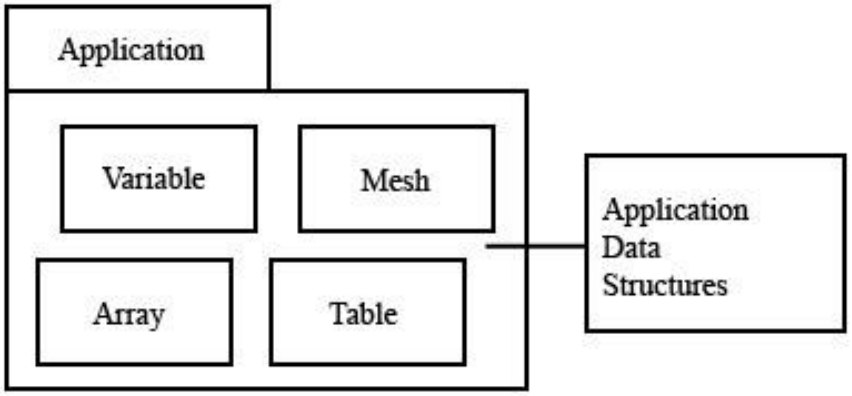
Data structure

In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.
Wikipedia

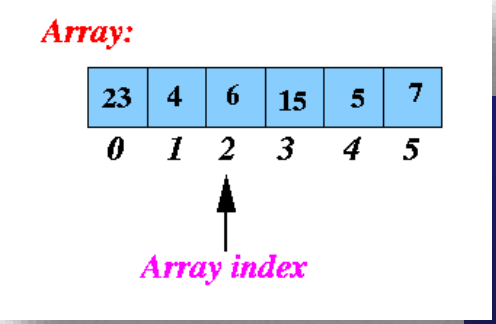


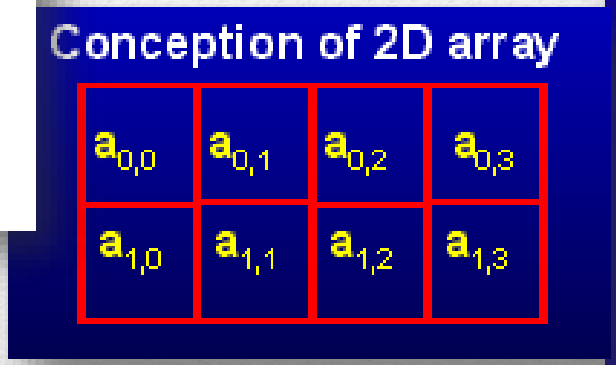
Recall this Definition

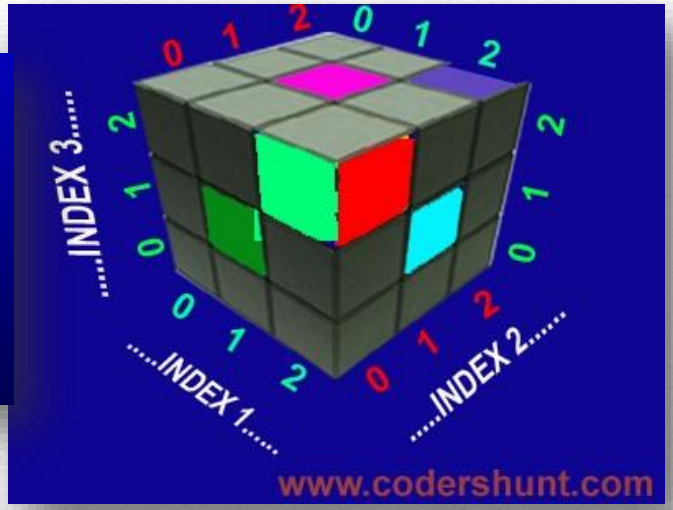
What is the difference between a *complex data type* and a *data structure*?



Question

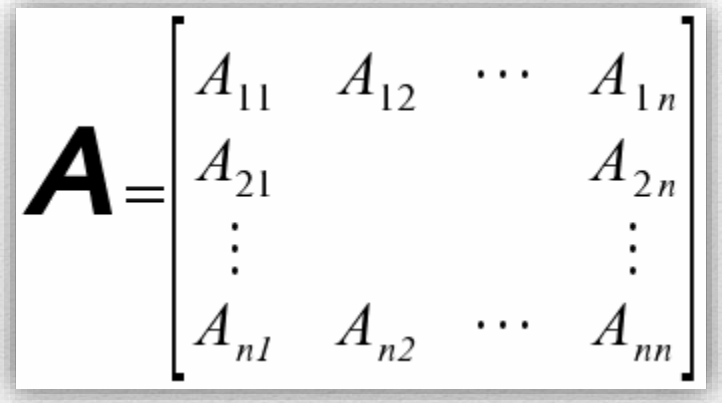






- An *array* is an arrangements of objects that follow a specific pattern (usually in rows, columns, *etc*.).

Array Example



- An *array* is an arrangements of objects that follow a specific pattern (usually in rows, columns, *etc*.).
- For example, this *complex data type* above is called a matrix and forms the foundation of *matrix algebra* that can be part of a mathematical algorithm.

Array Example

FACT:

An array can be looked upon as both a *complex data type* and a *data structure*.

Array Example

Array data structure - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Array_data_structure**
The simplest **type** of **data structure** is a linear **array**. This is also called one-dimensional **array**. In computer science, an **array data structure** or simply an **array** is a ...

Array data type - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Array_data_type**
In computer science, an **array** type is a **data type** that is meant to describe a ... **Array** types are often implemented by **array data structures**, but sometimes by ...

Array Example

- A *data type* is something "**visible**" to the programmer (*i.e.*, a matrix in a mathematical code).

Array Example

- A *data type* is something "**visible**" to the programmer (*i.e.*, a matrix in a mathematical code).
- A *data structure* is something "**invisible**" to the programmer, implemented behind the scenes (*i.e.*, the way the computer stores and manipulates, say, matrix data during program execution for the sake of efficiency).

Array Example

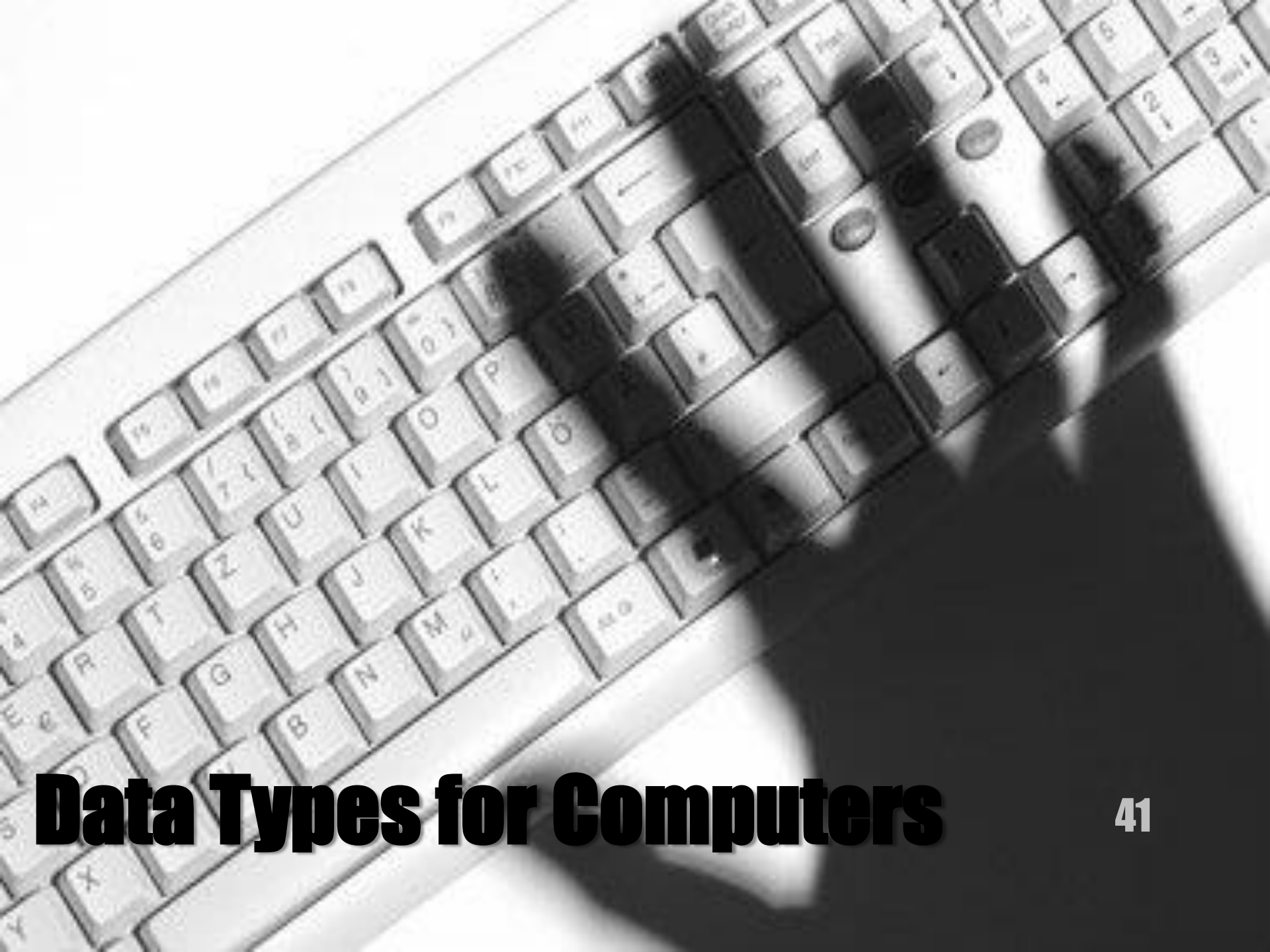
- A *data type* is something "**visible**" to the programmer (*i.e.*, a matrix in a mathematical code).
- A *data structure* is something "**invisible**" to the programmer, implemented behind the scenes (*i.e.*, the way the computer stores and manipulates, say, matrix data during program execution for the sake of efficiency).
- Thus, an array data type such as a matrix in a computer program may be efficiently implemented by an array data structure by a computer.

Array Example

- A *data type* is something "**visible**" to the programmer (*i.e.*, a matrix in a mathematical code).
- A *data structure* is something "**invisible**" to the programmer, implemented behind the scenes (*i.e.*, the way the computer stores and manipulates, say, matrix data during program execution for the sake of efficiency).
- Thus, an array data type such as a matrix in a computer program may be efficiently implemented by an array data structure by a computer.
- We will discuss this in more depth later in the course.

Array Example

End of Theory of Data



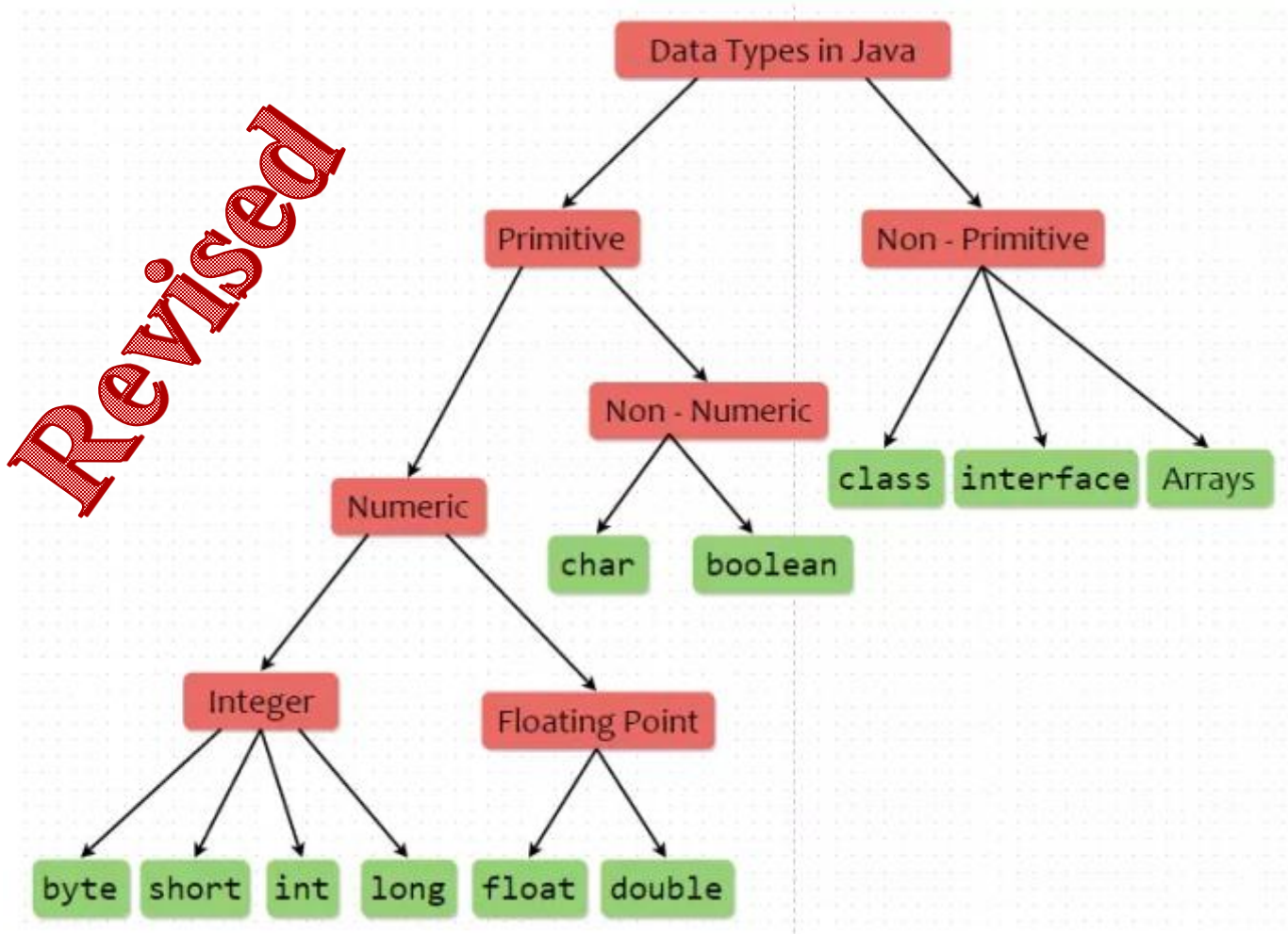
Data Types for Computers

First…

One difficulty for new programmers is figuring out how to represent data and information in order to describe the "real world" in a computer since the choices are so limited – the problem of determining a good abstraction.

Second…

This leads us to what we have been introducing, which is the *theory of data classes*.



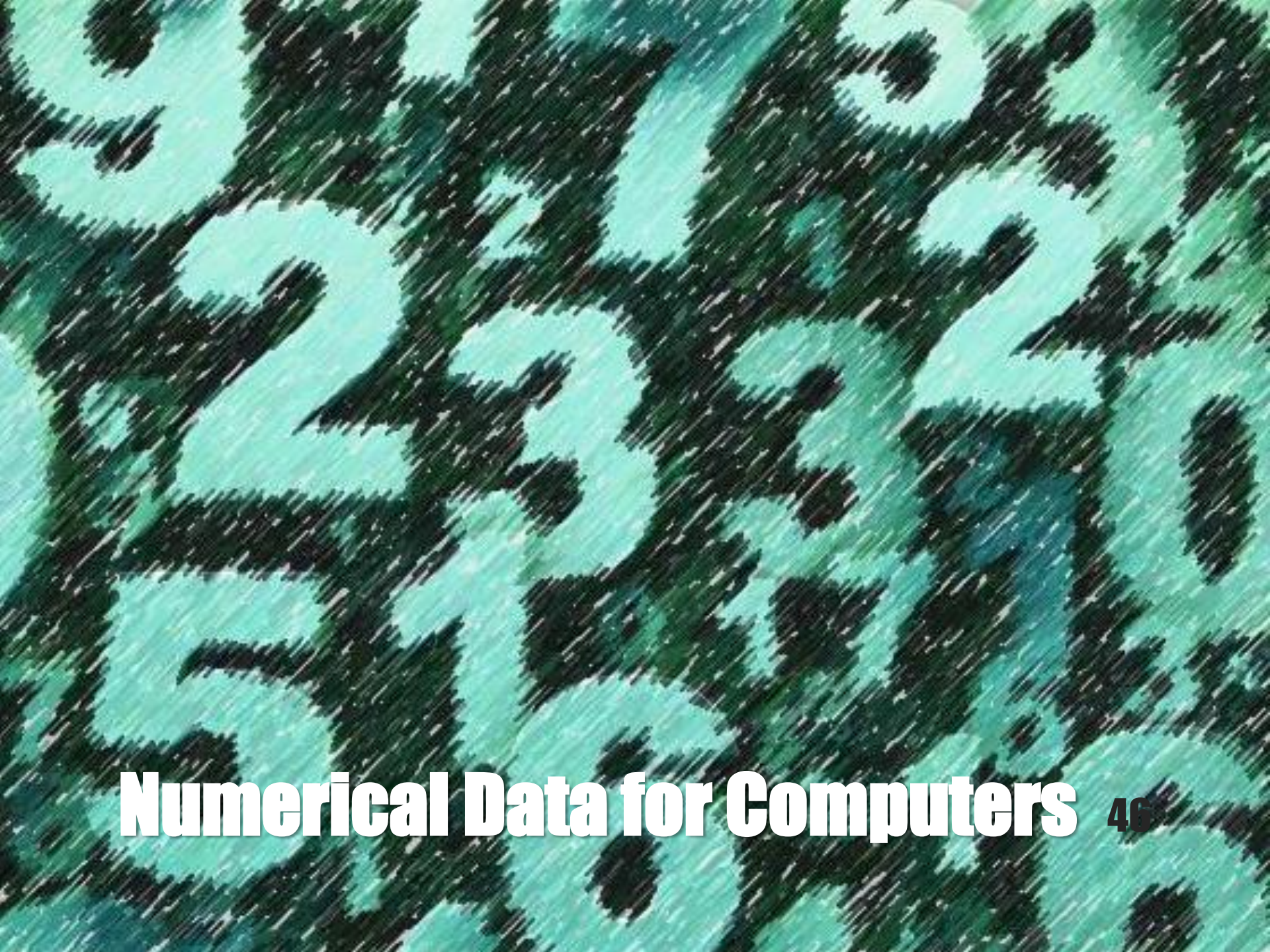
Example: Java Taxonomy

http://theoryofprogramming.com/2015/05/27/java-tutorials-data-types-input-operators-in-java/

Another Classification of Data

One can broadly classify data as either numeric or non-numeric. Examples of non-numerical data are:

- Characters,
- Strings (concatenation of characters),
- Booleans (True, False),
- *etc.*


Numerical Data for Computers
46

- Recall that the Central Processing Unit (CPU) contains the Arithmetic/Logic Unit (ALU – the circuit that performs the basic arithmetic and logic operations) and the CU (traffic cop).

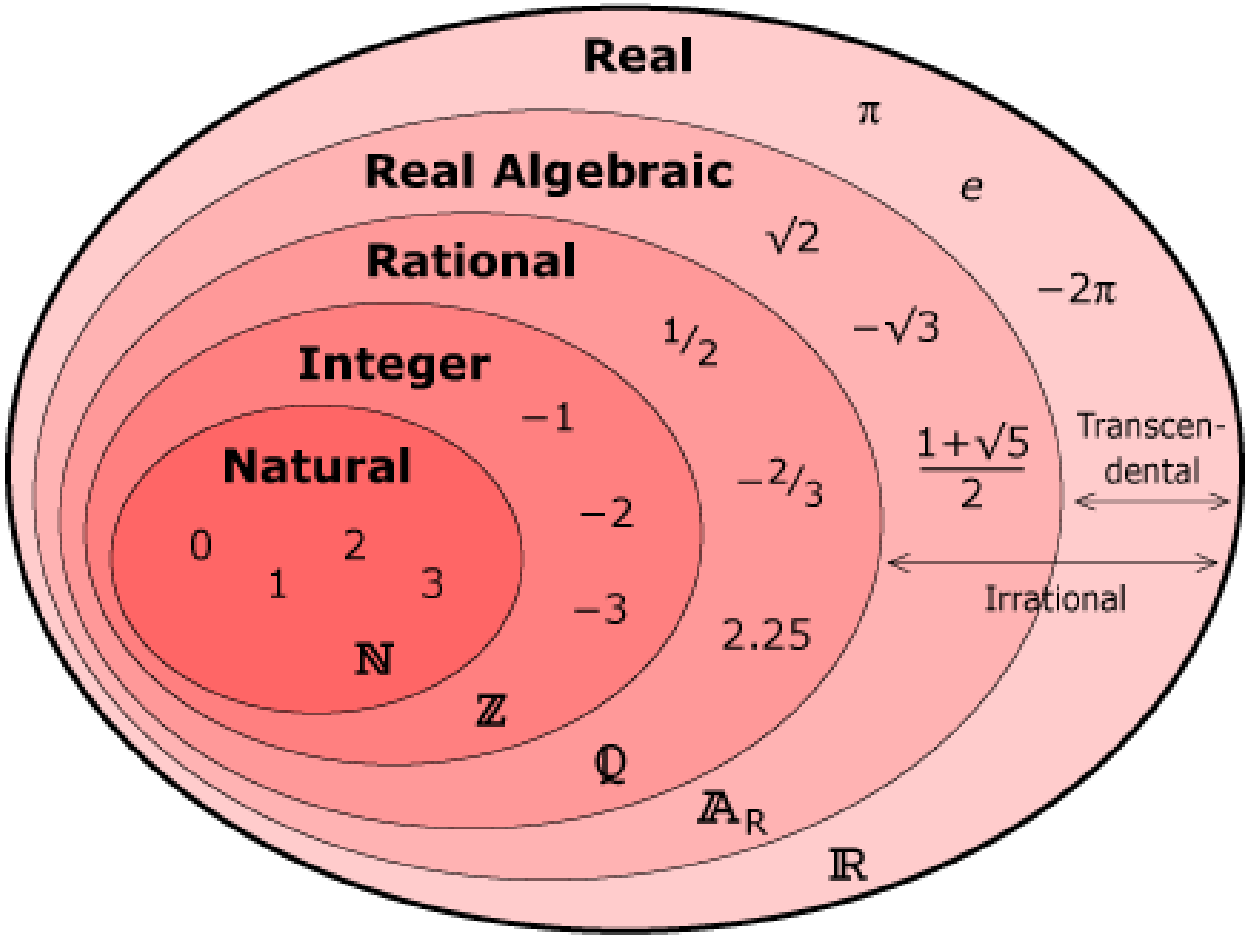
Numerical Data for Computers

- Recall that the Central Processing Unit (CPU) contains the Arithmetic/Logic Unit (ALU – the circuit that performs the basic arithmetic and logic operations) and the CU (traffic cop).
- A computer program mostly processes numerical and logical data into useful information of one sort or another, in part, through these circuits.

Numerical Data for Computers

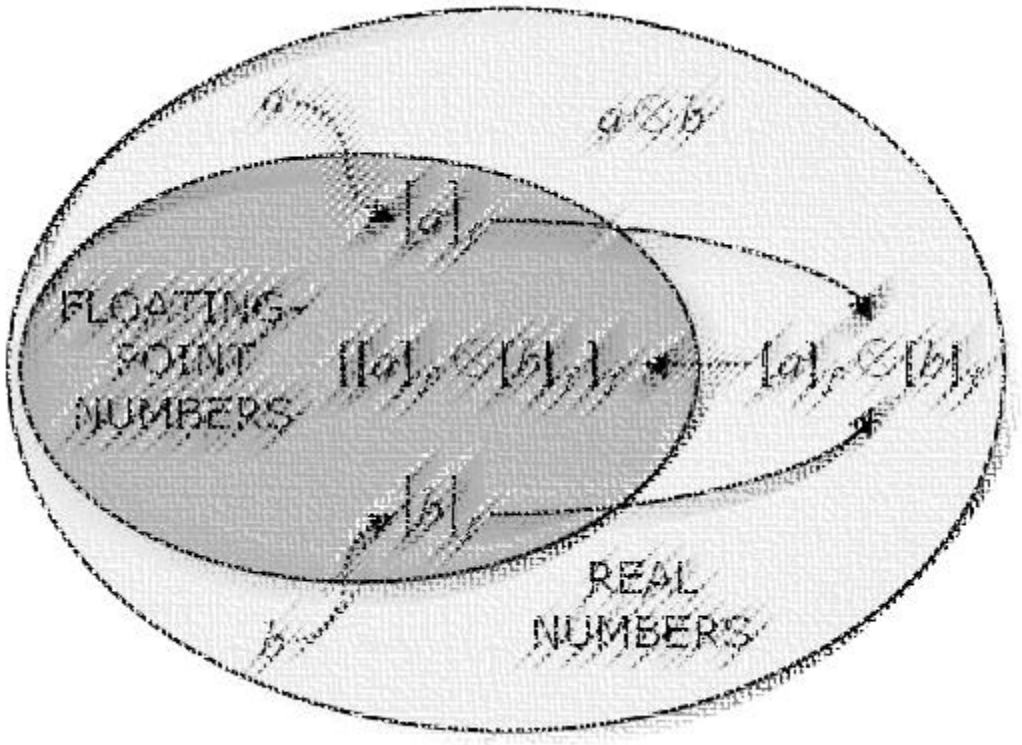
- Recall that the Central Processing Unit (CPU) contains the Arithmetic/Logic Unit (ALU – the circuit that performs the basic arithmetic and logic operations) and the CU (traffic cop).
- A computer program mostly processes numerical and logical data into useful information of one sort or another, in part, through these circuits.
- Von Neumann computers are binary and, therefore, *integer in nature*.

Numerical Data for Computers



Common Number Sets

http://thinkzone.wlonk.com/Numbers/NumberSets.htm



Floating Point Numbers

- Clearly, real numbers are more challenging to represent in a computer than integers.

Floating Point Numbers

- Clearly, real numbers are more challenging to represent in a computer than integers.
- We represent real numbers on computers as **floating point decimal numbers**:
  - The term, floating point, means there are no fixed number of digits before and after the decimal point (*e.g.*, the decimal point can *float*).

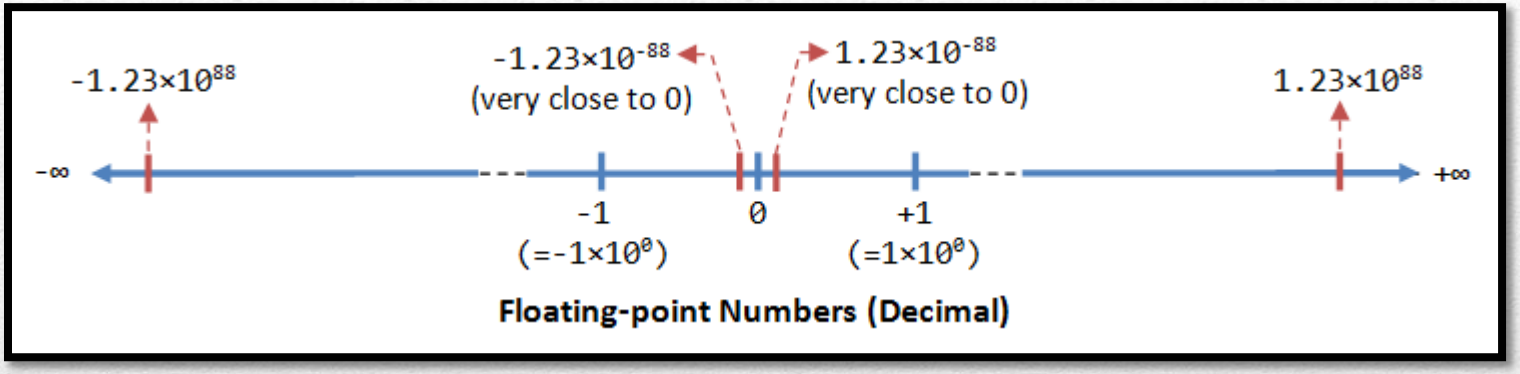
Floating Point Numbers

- Clearly, real numbers are more challenging to represent in a computer than integers.
- We represent real numbers on computers as **floating point decimal numbers**:
  - The term, floating point, means there are no fixed number of digits before and after the decimal point (*e.g.*, the decimal point can *float*).
- Note that real numbers require more computing power to process them – some computers have a *floating point unit* chip (FPU) dedicated to this task.

Floating Point Numbers

Floating Point Numbers

- Clearly, real numbers are more challenging to represent in a computer than integers.
- We represent real numbers on computers as **floating point decimal numbers:**
  - The term, floating point, means there are no fixed number of digits before and after the decimal point (*e.g.*, the decimal point can *float*).
- Note that real numbers require more computing power to process them – some computers have a *floating point unit* chip (FPU) dedicated to this task.
- **Most floating point numbers are only *approximations* when represented in a computer (good to $1:10^{15}$).**



Since floating point numbers are approximations, the set of such numbers is a proper subset of real numbers since not all real numbers are representable on a computer.

Floating Point Numbers

Statement: The number, one, is represented by the symbol, 1.

Question

Statement: The number, one, is represented by the symbol, 1.

Question: What is the data type of the second-to-last character in this statement?

Question

- Numbers can be numeric as in an arithmetic expression, or characters as in the previous statement.

Answer

- Numbers can be numeric as in an arithmetic expression, or characters as in the previous statement.
- There are Java *functions* for conversion from one data type to another.

Answer

- Numbers can be numeric as in an arithmetic expression, or characters as in the previous statement.
- There are Java *functions* for conversion from one data type to another.
- Clearly, operators include *functions* (later topic).

Here are some quick reference example codes:

- Convert using Integer.toString(int)
- Convert using String.valueOf(int)
- Convert using new Integer(int).toString()
- Convert using String.format()
- Convert using DecimalFormat
- Convert using StringBuffer or StringBuilder
- Quick Solution
- Convert with special radix (not base 10 system)

JavaDevNotes.com

Answer

`Integer.toString` calls the static method in the class `Integer`. It does not need an instance of `Integer`.

If you call `new Integer(i)` you create an instance of type `Integer`, which is a full Java object encapsulating the value of your int. Then you call the `toString` method on it to ask it to return a string representation of *itself*.

If all you want is to print an `int`, you'd use the first one because it's lighter, faster and doesn't use extra memory (aside from the returned string).

OO Details of This Example

Non-Numerical Data Types

Boolean Data Types

True and **False** are Java's Boolean variables.

Boolean Data Types


String Data Types

String (computer science) - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/String_(computer_science)
In **computer** programming, a **string** is traditionally a sequence of characters, either as a ... For any two **strings** s and t in Σ*, their concatenation is **defined** as the ...
Formal theory - String datatypes - Text file strings - Non-text strings

Example: A byte is an 8-bit binary string (*i.e.*, 10110110)

Definition

Creating Strings

The most direct way to create a string is to write –

```
String greeting = "Hello world!";
```

Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

As with any other object, you can create String objects by using the new keyword and a constructor. The String class has 11 constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.

Java String Syntax

Creating Strings

The most direct way to create a string is to write –

```
String greeting = "Hello world!";
```

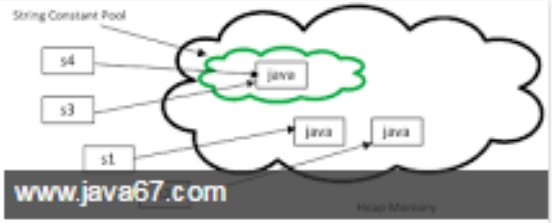
Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

As with any other object, you can create String objects by using the new keyword and a constructor. The String class has 11 constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.

NOTE: A string object is a **variable**; a string literal is a **constant** (a fixed sequence of characters between quotation marks).

Java String Syntax

A **String** object is an individual instance of the **java**.lang.**String** class. ... This means, that the character sequence "abcde" will be stored at a central place, and whenever the same **literal** "abcde" is used again, the JVM will not create a new **String** object but use the reference of the cached **String**. Jul 21, 2010



java - Difference between string object and string literal - Stack Overflow
stackoverflow.com/questions/3297867/difference-between-string-object-and-string-literal

A String *literal* is a String *object*, but a String *object* is not necessarily a String *literal*. And once assigned to a reference variable, it's all but impossible to tell if a given String object is a *literal* or not.

String vs. String Literal

A **Java virtual machine** (**JVM**) is an abstract computing machine that enables a computer to run a Java program. There are three notions of the **JVM**: specification, implementation, and instance. The specification is a document that formally describes what is required of a **JVM** implementation.

Java virtual machine - Wikipedia
https://en.wikipedia.org/wiki/Java_virtual_machine

NOTE: JVM

Example

```
public class StringDemo {

   public static void main(String args[]) {
      char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
      String helloString = new String(helloArray);
      System.out.println( helloString );
   }
}
```

This will produce the following result –

Output

```
hello.
```

Java String Syntax

- Composite types are derived from more than one primitive type. This can be done in a number of ways. The ways they are combined are called data structures. Composing a primitive type into a compound type generally results in a new type, e.g. *array-of-integer* is a different type to *integer*.
- Common examples include *Arrays*, *Records*, *Unions*, *Sets and Objects*.

Composite Data Types

- An **array** stores a number of elements of the same type in a specific order. They are accessed randomly using an integer to specify which element is required (although the elements may be of almost any type). Arrays may be fixed-length or expandable.
- A **list** is similar to an array, but its contents are strung together by a series of references to the next element.

Composite Data Types

- **Record** (also called tuple or struct) are among the simplest data structures. A record is a value that contains other values, typically in fixed number and sequence and typically indexed by names. The elements of records are usually called *fields* or *members*.

Composite Data Types

- A **union** type definition will specify which of a number of permitted primitive types may be stored in its instances, e.g. "float or long integer". Contrast with a record, which could be defined to contain a float *and* an integer; whereas, in a union, there is only one type allowed at a time.

Composite Data Types

- A **union** type definition will specify which of a number of permitted primitive types may be stored in its instances, e.g. "float or long integer". Contrast with a record, which could be defined to contain a float *and* an integer; whereas, in a union, there is only one type allowed at a time.
- A **tagged union** (also called a variant, variant record, discriminated union, or disjoint union) contains an additional field indicating its current type, for enhanced type safety.

Composite Data Types

- A **set** is an abstract data structure that can store certain values, without any particular order, and no repeated values. Values themselves are not retrieved from sets, rather one tests a value for membership to obtain a boolean "in" or "not in".

Composite Data Types

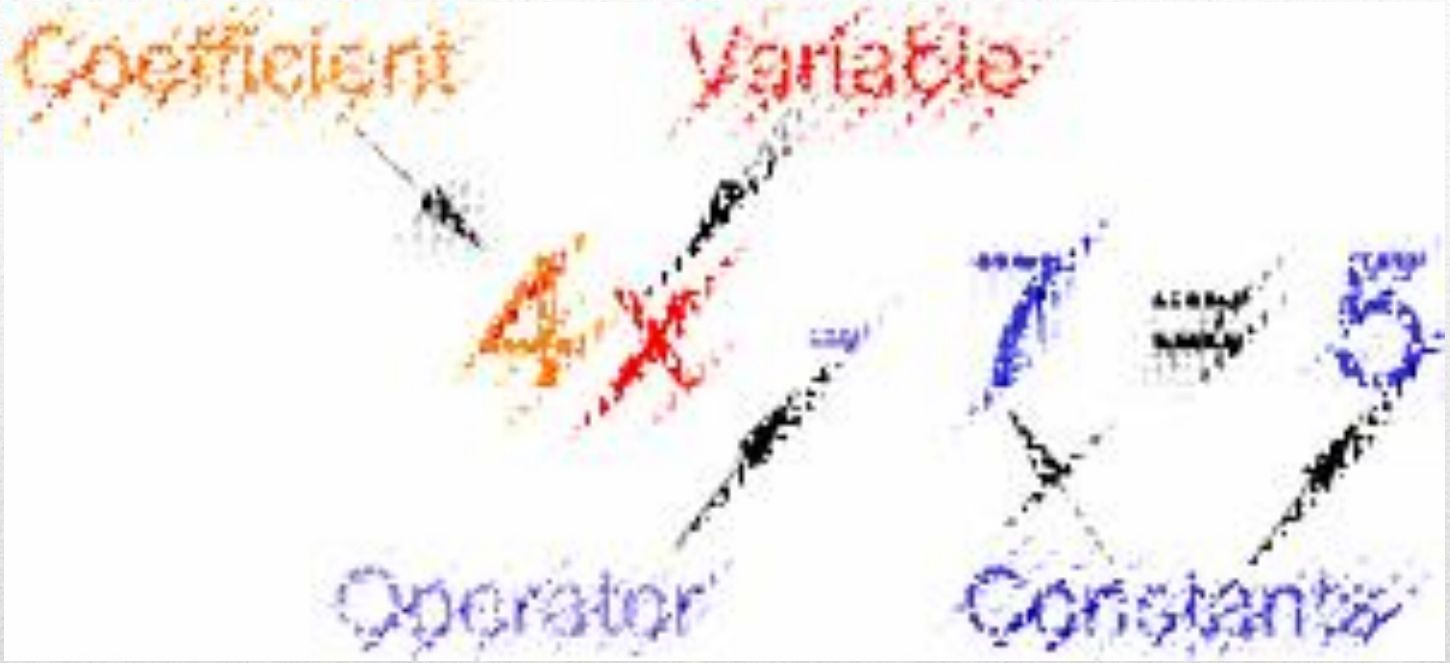
- An **object** contains a number of data fields, like a record, and also a number of subroutines for accessing or modifying them, called *methods*.

Composite Data Types

Pointers and references [edit]

The main non-composite, derived type is the pointer, a data type whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address. It is a primitive kind of reference. (In everyday terms, a page number in a book could be considered a piece of data that refers to another one). Pointers are often stored in a format similar to an integer; however, attempting to dereference or "look up" a pointer whose value was never a valid memory address would cause a program to crash. To ameliorate this potential problem, pointers are considered a separate type to the type of data they point to, even if the underlying representation is the same.

Other Important Data Types



Constants and Variables

- Let us reconsider two basic “well known” functions:
  - Monomial: x^n
  - Exponential: n^x

Constants and Variables

Constants and Variables

- Let us reconsider two basic “well known” functions:
  - Monomial: x^n
  - Exponential: n^x
- They are composed of two fundamental units – a *constant* and a *variable*.

- In mathematics, a constant is the term used to describe a parameter that never varies such as the number π.

Constants and Variables

- In mathematics, a constant is the term used to describe a parameter that never varies such as the number π.
- An unspecified constant is given an alphabetical character and is usually assigned from the beginning of the alphabet, such as "*a*".

Constants and Variables

- In mathematics, a constant is the term used to describe a parameter that never varies such as the number π.
- An unspecified constant is given an alphabetical character and is usually assigned from the beginning of the alphabet, such as "*a*".
- On the other hand, a variable is an alphabetical character that represents a number that is unspecified or unknown and is usually assigned from the end of the alphabet, such as "*x*".

Constants and Variables

- In mathematics, a constant is the term used to describe a parameter that never varies such as the number π.
- An unspecified constant is given an alphabetical character and is usually assigned from the beginning of the alphabet, such as "*a*".
- On the other hand, a variable is an alphabetical character that represents a number that is unspecified or unknown and is usually assigned from the end of the alphabet, such as "*x*".
- For example, *a*, *b*, and *c* are all constants whereas *x* is unknown in the famous *expression* known as the quadratic equation:

$$ax^2 + bx + c = 0$$

Constants and Variables

Definition: Expression (CS)

A statement combining values, constants, variables, operators, and functions, interpreted according to the syntax of the programming language at issue, which computes and then produces an *evaluation* (numerical, string, logical, *etc*.)

Expression

For most programming languages:

- A program constant is an immediate value found in an expression.

Variables in Programming

Variables in Programming

For most programming languages:

- A program constant is an immediate value found in an expression.
- A program variable is a **memory location**, associated with a symbolic name (an *identifier*), which contains some known or unknown information.

Variables in Programming

For most programming languages:

- A program constant is an immediate value found in an expression.
- A program variable is a **memory location**, associated with a symbolic name (an *identifier*), which contains some known or unknown information.
- The value of the variable may change during execution (for example, by setting, retrieving and updating data).

Variables in Programming

For most programming languages:

- A program constant is an immediate value found in an expression.
- A program variable is a **memory location**, associated with a symbolic name (an *identifier*), which contains some known or unknown information.
- The value of the variable may change during execution (for example, by setting, retrieving and updating data).

These definitions do not necessarily correspond to that given in mathematics insofar as a computing variable may not be part of an equation or formula as is always the case with a mathematical variable.

The Java programming language defines the following kinds of variables:

- **Instance Variables (Non-Static Fields)** Technically speaking, objects store their individual states in "non-static fields", that is, fields declared without the `static` keyword. Non-static fields are also known as *instance variables* because their values are unique to each *instance* of a class (to each object, in other words); the `currentSpeed` of one bicycle is independent from the `currentSpeed` of another.
- **Class Variables (Static Fields)** A *class variable* is any field declared with the `static` modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated. A field defining the number of gears for a particular kind of bicycle could be marked as `static` since conceptually the same number of gears will apply to all instances. The code `static int numGears = 6;` would create such a static field. Additionally, the keyword `final` could be added to indicate that the number of gears will never change.
- **Local Variables** Similar to how an object stores its state in fields, a method will often store its temporary state in *local variables*. The syntax for declaring a local variable is similar to declaring a field (for example, `int count = 0;`). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared — which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.
- **Parameters** You've already seen examples of parameters, both in the `Bicycle` class and in the `main` method of the "Hello World!" application. Recall that the signature for the `main` method is `public static void main(String[] args)`. Here, the `args` variable is the parameter to this method. The important thing to remember is that parameters are always classified as "variables" not "fields". This applies to other parameter-accepting constructs as well (such as constructors and exception handlers) that you'll learn about later in the tutorial.

Variables in Java

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

- All compile-based programming languages require **declaration** *statements* which announce to the compiler the existence of language elements, such as the name and type of a variable used in a program (*i.e.*, REAL*8 is a global declaration statement in FORTAN that defines all real numbers to be of a certain precision).

Declaring/Assigning Variables 94

Declaring/Assigning Variables 95

- All compile-based programming languages require **declaration** *statements* which announce to the compiler the existence of language elements, such as the name and type of a variable used in a program (*i.e.*, REAL*8 is a global declaration statement in FORTAN that defines all real numbers to be of a certain precision).
- Java is a **strongly typed** programming language. This means that every variable must have a data type associated with it. For example, a variable could be declared to use one of the eight primitive data types: byte, short, int, long, float, double, char or boolean. To declare a variable in Java, just use the data type followed by the variable name (*i.e.*, int NumberOfDays;) ThoughtCo

The rules and conventions for **naming** your **variables** can be summarized as follows: **Variable names** are case-sensitive. A **variable's name** can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign " $ ", or the underscore character " _ ".

Variables (The Java™ Tutorials > Learning the Java Language ...
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Naming Variables in Java

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Naming

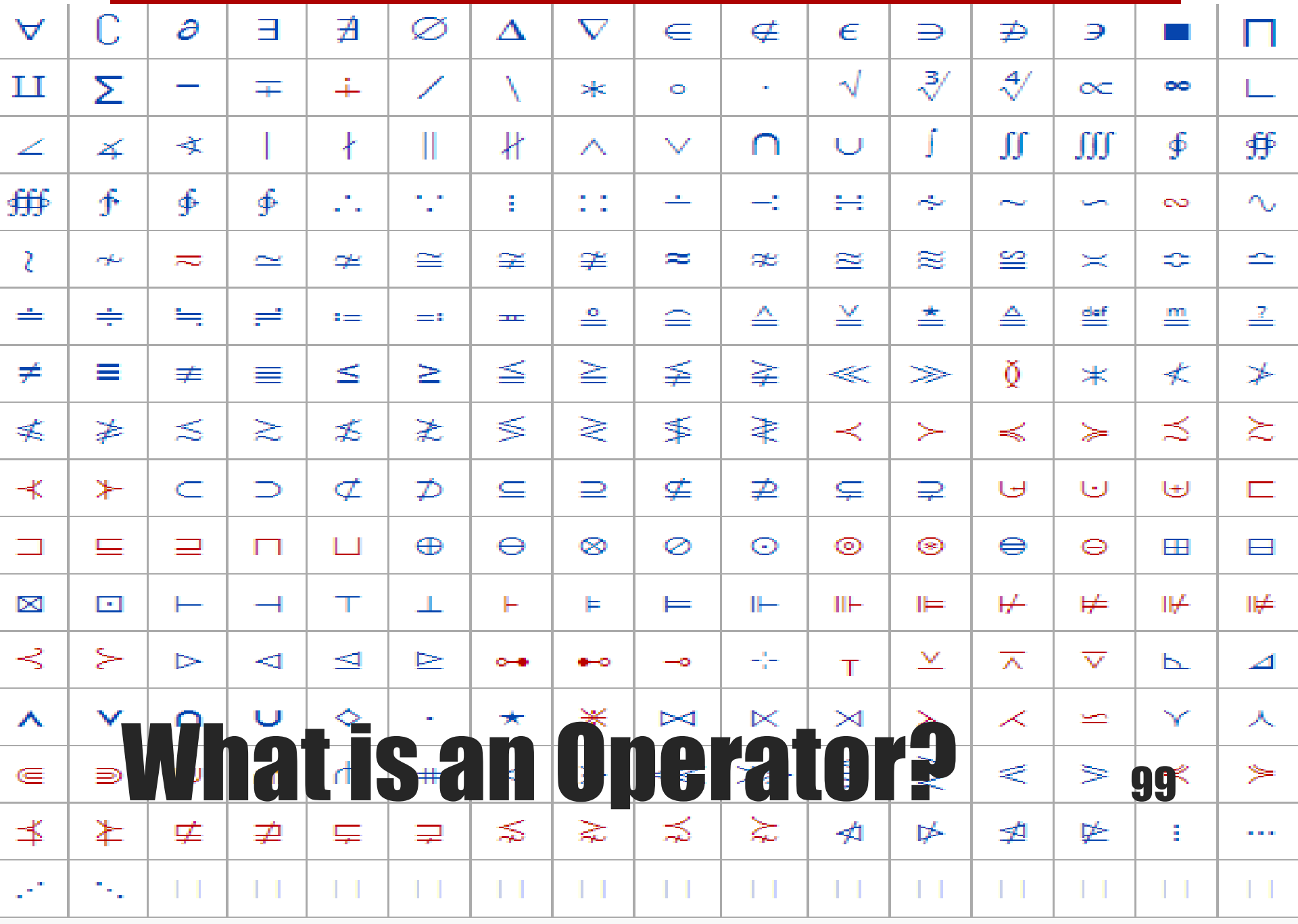
Every programming language has its own set of rules and conventions for the kinds of names that you're allowed to use, and the Java programming language is no different. The rules and conventions for naming your variables can be summarized as follows:

- Variable names are case-sensitive. A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "$", or the underscore character "_". The convention, however, is to always begin your variable names with a letter, not "$" or "_". Additionally, the dollar sign character, by convention, is never used at all. You may find some situations where auto-generated names will contain the dollar sign, but your variable names should always avoid using it. A similar convention exists for the underscore character; while it's technically legal to begin your variable's name with "_", this practice is discouraged. White space is not permitted.
- Subsequent characters may be letters, digits, dollar signs, or underscore characters. Conventions (and common sense) apply to this rule as well. When choosing a name for your variables, use full words instead of cryptic abbreviations. Doing so will make your code easier to read and understand. In many cases it will also make your code self-documenting; fields named `cadence`, `speed`, and `gear`, for example, are much more intuitive than abbreviated versions, such as `s`, `c`, and `g`. Also keep in mind that the name you choose must not be a keyword or reserved word.
- If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word. The names `gearRatio` and `currentGear` are prime examples of this convention. If your variable stores a constant value, such as `static final int NUM_GEARS = 6`, the convention changes slightly, capitalizing every letter and separating subsequent words with the underscore character. By convention, the underscore character is never used elsewhere.

Naming Variables in Java

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

OPERATORS


What is an Operator?

What is an Operator?

100

Definition:

An operator in a programming language is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and.
- Assignment Operator.

Operators in Java - Javatpoint
https://www.javatpoint.com/operators-in-java

JAVA OPERATORS

Highest	P	Parentheses	( )
	E	Exponentiation	**
	M	Multiplication	*
	D	Division	/
	A	Addition	+
Lowest	S	Subtraction	-

Priority of Arithmetic Operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Complete Set of Java Numeric Operators

https://www.tutorialspoint.com/java/java_basic_operators.htm

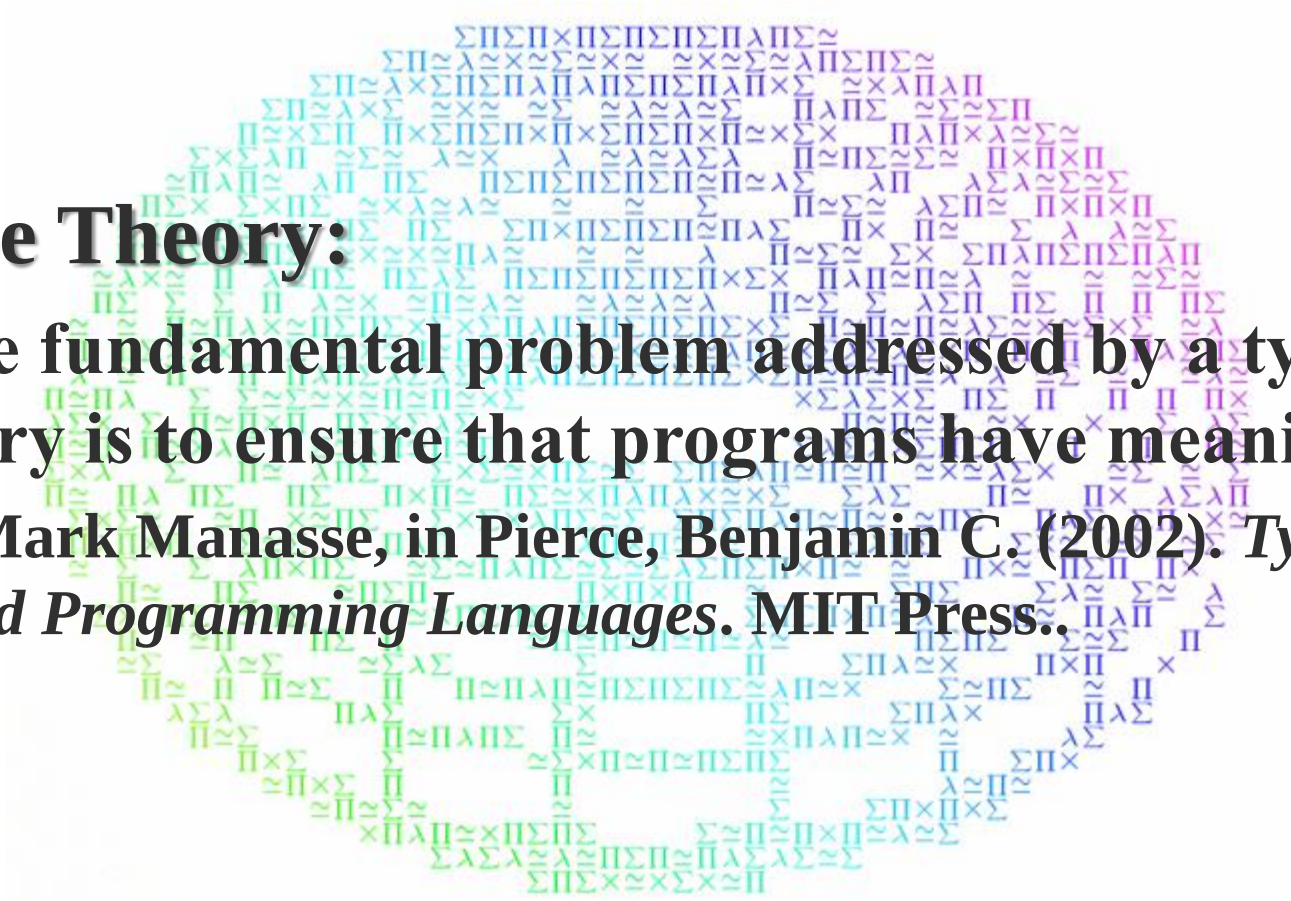
Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr ~ !*
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

JAVA OPERATOR PRECEDENCE

javaTpoint

https://www.javatpoint.com/operators-in-java

End Operator Unit



Type Theory:

"The fundamental problem addressed by a type theory is to ensure that programs have meaning."

- Mark Manasse, in Pierce, Benjamin C. (2002). *Types and Programming Languages*. MIT Press..

Recall Type Theory

Typing

Typing assigns a data type, giving meaning to a sequence of bits such as a value in memory or some object such as a variable.

Definition

- The hardware of a general purpose computer is unable to discriminate between, for example, a memory address and an instruction code, or between a character, an integer, or a floating-point number, because it makes no intrinsic distinction between any of the possible values that a sequence of bits might *mean*.

Typing

- The hardware of a general purpose computer is unable to discriminate between, for example, a memory address and an instruction code, or between a character, an integer, or a floating-point number, because it makes no intrinsic distinction between any of the possible values that a sequence of bits might *mean*.
- Associating a sequence of bits with a type conveys that meaning to the programmable hardware to form a *symbolic system* composed of that hardware and some program.

Typing

Type Checking

The process of verifying and enforcing the constraints of types. This may be implemented either at compile-time (a static check) or run-time (a dynamic check).

Definition

A programming language is said to be **dynamically** typed, or just '**dynamic**', when the majority of its **type** checking is performed at run-time as opposed to at compile-time. In **dynamic typing**, **types** are associated with values not variables. Oct 4, 2009

static typing - What is the difference between statically typed and ...
stackoverflow.com/.../what-is-the-difference-between-statically-typed-and-dynamically-t...

Dynamic vs. Static Type Checking

Java is a *statically typed* language, so the compiler does most of this checking for you. Once you declare a variable to be a certain type, the compiler will ensure that it is only ever assigned values of that type (or values that are sub-types of that type).

Dynamic vs. Static Type Checking

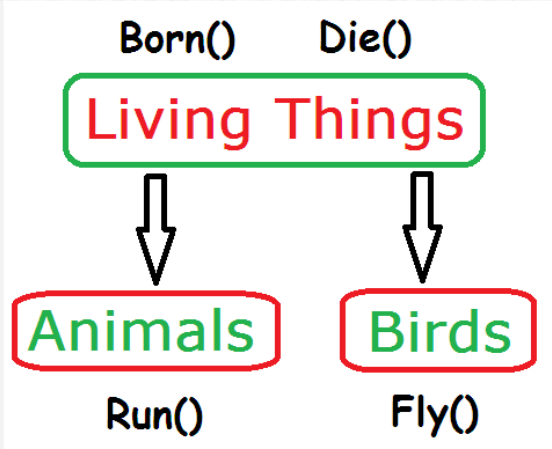
Upcasting & Downcasting

Upcasting is basically an object creation mechanism in which we create objects by referring to their base class. We do this by replacing the sub-class by the base-class in the object definition. This particularly comes in handy when you know that a specialized object created will not necessarily use all the functions that it has got to offer. So, replace a subclass(inherited class) by a base-class and all you have done is Upcasting.

Upcasting & Downcasting

http://www.dotnetstation.com/CSharpWinForms/upcasting-and-downcasting-in-csharp-5

This concept can be well understood if we take an example. Lets suppose we have three classes. One parent or generalized class called the `LIVING_THINGS` class and the two subclasses `ANIMAL` and `BIRD` which inherit from the former. The parent class or the baseclass has the function, say `Born()` and `Die()`. The specialized class have the function, say `Run()` and `Fly()` for `ANIMAL` and `BIRD` respectively.



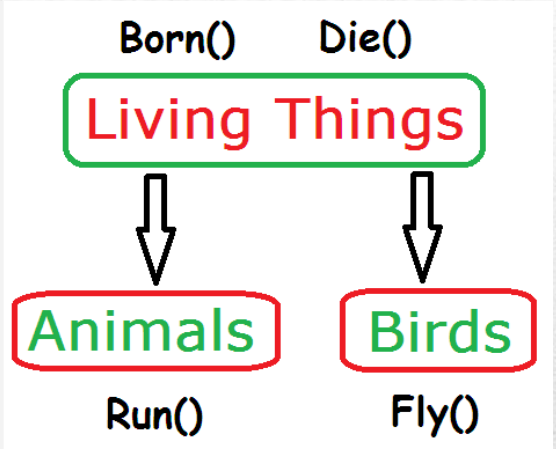
Upcasting & Downcasting

http://www.dotnetstation.com/CSharpWinForms/upcasting-and-downcasting-in-csharp-5

To create an ANIMAL and BIRD object you would usually use the syntax below:

- `ANIMAL animal = new ANIMAL();`
- `BIRD bird = new BIRD();`

In the code above, the methods `Run()` and `Fly()` work perfectly with the objects created. This is the normal way in which you would create the objects. The reference used to create the object is exactly the same as the object type created during run-time. The methods of the subclass work just fine.



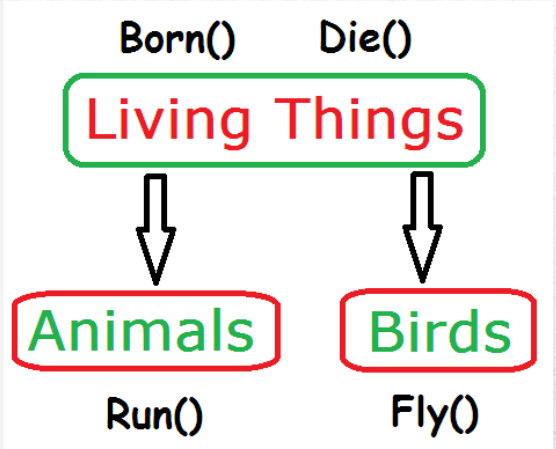
Upcasting & Downcasting

*http://www.dotnetstation.com/CSharpWinForms/upcasting-and-downcasting-in-**csharp**-5*

But, to create an Upcast, you would use the syntax below:

- `LIVING_THINGS animal = new ANIMAL();`

In the code above, even though we have created an object of the type ANIMAL and BIRD, the reference used is actually of the base-class. Therefore the methods `Run()` and `Fly()` are not available to these objects. Instead only the functions available in the base-class such as `Born()` and `Die()` are available.
This can be useful if you want only basic functions available at the start and want to later convert these objects of their original types to make their specialized functions available afterwards. This can be done with the help of *Downcasting*. Downcasting can change the reference types of these objects to the specialized subclass to make available the functions which were not available otherwise. Watch the code below:



Upcasting & Downcasting

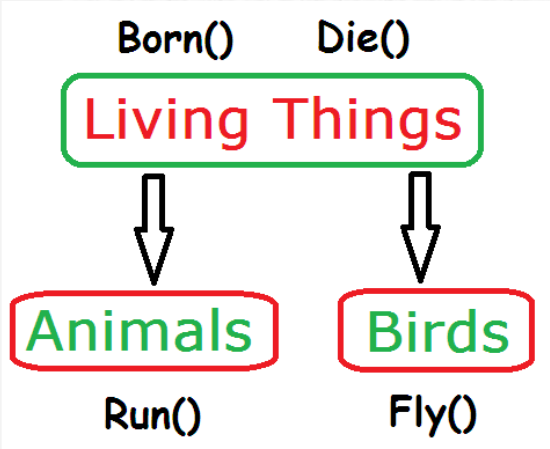
http://www.dotnetstation.com/CSharpWinForms/upcasting-and-downcasting-in-csharp-5

- `ANIMAL new_animal = animal as ANIMAL;`

In the above code, we have created a new object and converted it to an ANIMAL type. It should be a good practice to actually check if the object to be converted supports the conversion. This can be done with the help o the *is* keyword

```
if (animal is ANIMAL)
{
    ANIMAL new_animal = animal as ANIMAL;
    new_animal.Run();
}
```

With the help of Downcasting, we have restored all the functions and properties that the `ANIMAL` object should have. Now it behaves as a proper `ANIMAL` object.

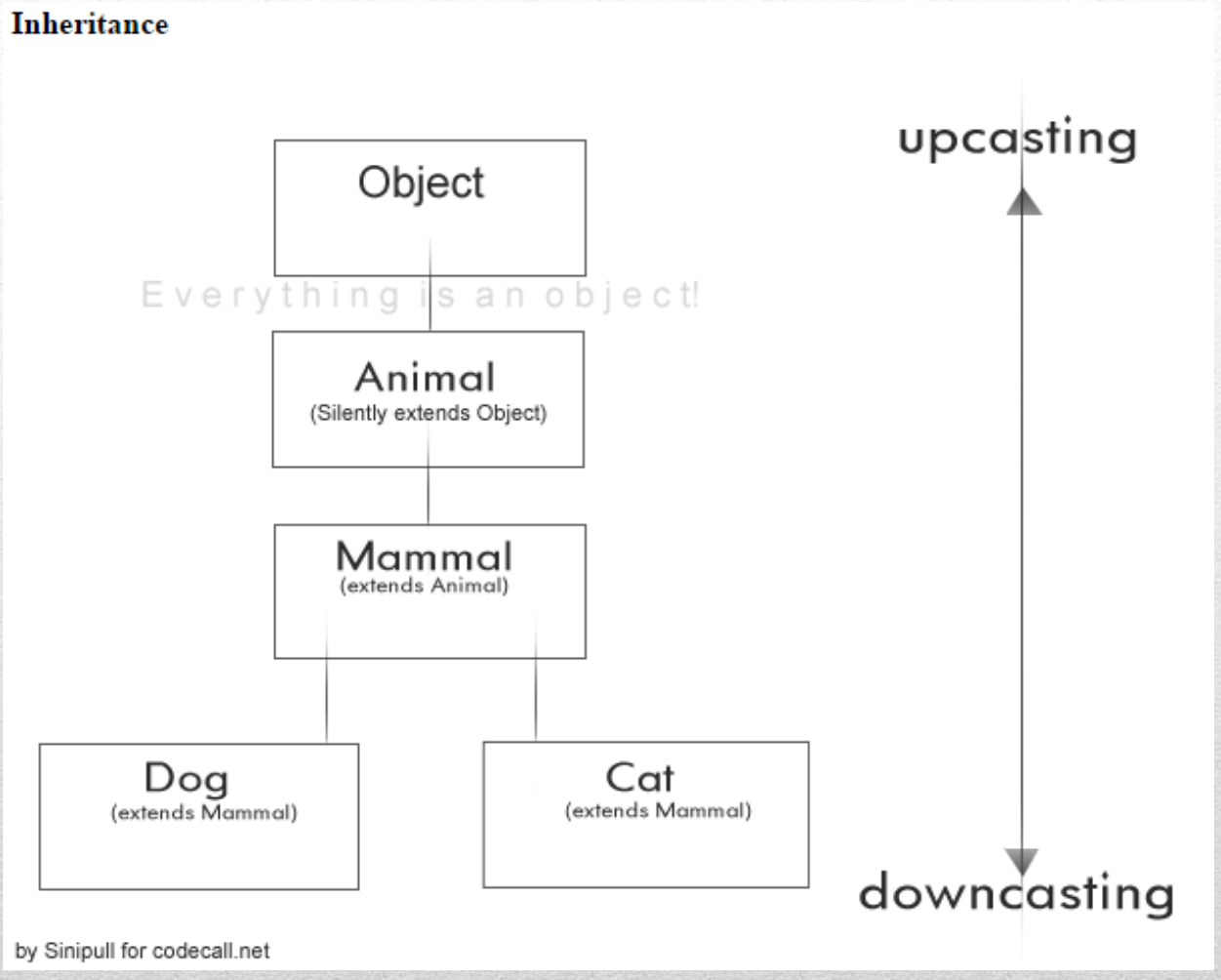


Upcasting & Downcasting

http://www.dotnetstation.com/CSharpWinForms/upcasting-and-downcasting-in-csharp-5

Summary

- **Upcasting** and **downcasting** are important aspects of Java, which allow us to build complicated programs using simple syntax, and gives us great advantages, like Polymorphism or grouping different objects together . Java permits an object of a subclass type to be treated as an object of any superclass type. *This is called upcasting*.
- Upcasting is done automatically, while downcasting must be manually done by the programmer.
- Upcasting and downcasting are **NOT** like casting primitives from one to other, and this causes a lot of confusion.

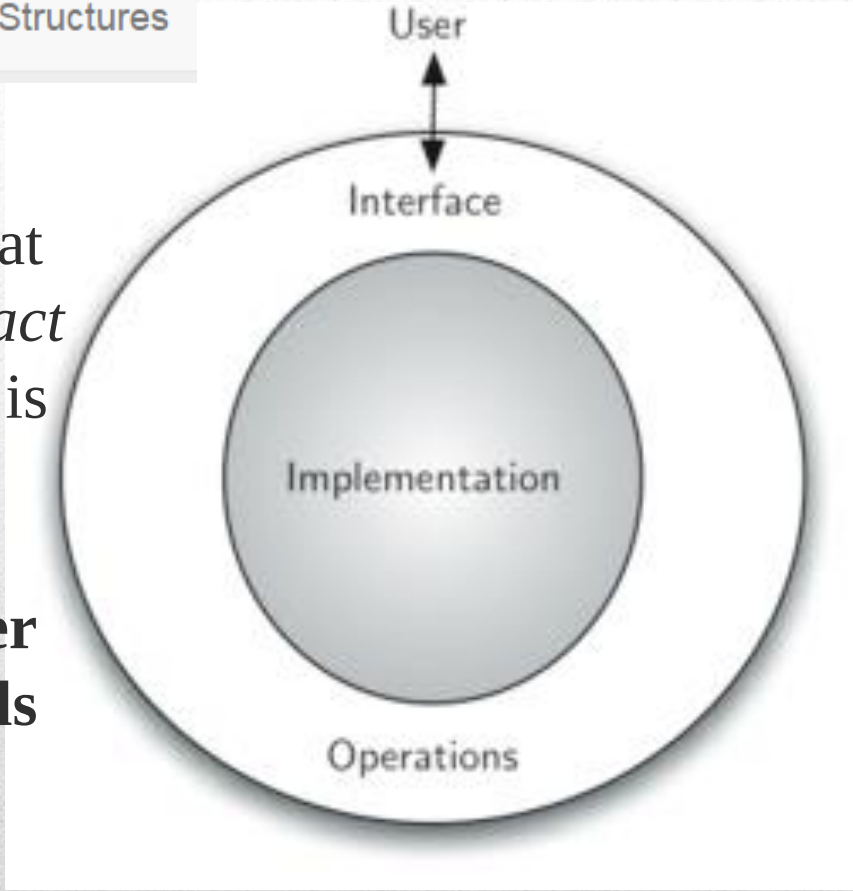


Upcasting and Downcasting in Java

http://www.cs.utexas.edu/~cannata/cs345

Abstract Data Types

The user interacts with the interface, using the operations that have been specified by the *abstract data type*. The abstract data type is the shell that the user interacts with. The implementation is hidden one level deeper. **The user is not concerned with the details of the implementation**.



Abstract Data Types

Any type that does not specify an implementation is an abstract data type. For instance, a stack (which is an abstract type) can be implemented as an array (a contiguous block of memory containing multiple values), or as a linked list (a set of non-contiguous memory blocks linked by pointers).

Abstract types can be handled by code that does not know or "care" what underlying types are contained in them. Programming that is agnostic about concrete data types is called generic programming. Arrays and records can also contain underlying types, but are considered concrete because they specify how their contents or elements are laid out in memory.

Abstract Data Types

Examples include:

- A queue is a first-in first-out list. Variations are Deque and Priority queue.
- A set can store certain values, without any particular order, and with no repeated values.
- A stack is a last-in, first out data structure.
- A tree is a hierarchical structure.
- A graph.
- A hash, dictionary, map or associative array is a more flexible variation on a record, in which name-value pairs can be added and deleted freely.
- A smart pointer is the abstract counterpart to a pointer. Both are kinds of references.

Abstract Data Types

> A data type or simply type is a classification identifying one of various types of data, such as real-valued, integer or Boolean, that determines the possible values for that type; the operations that can be done on values of that type; the meaning of the data; and the way values of that type can be stored.

Now, as per Wikipedia, there are various definitions for "type" in data type.

The question you have asked is a good one. There are data types in today's modern languages, that are referred to as **Abstract Data Types** or ADT in short. The definition of an ADT is:

> An abstract data type (ADT) is a mathematical model for a certain class of data structures that have similar behavior; or for certain data types of one or more programming languages that have similar semantics. An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Basic Summary

It is also written that:

> Abstract data types are purely theoretical entities, used (among other things) to simplify the description of abstract algorithms, to classify and evaluate data structures, and to formally describe the type systems of programming languages. However, an ADT may be implemented by specific data types or data structures, in many ways and in many programming languages; or described in a formal specification language.

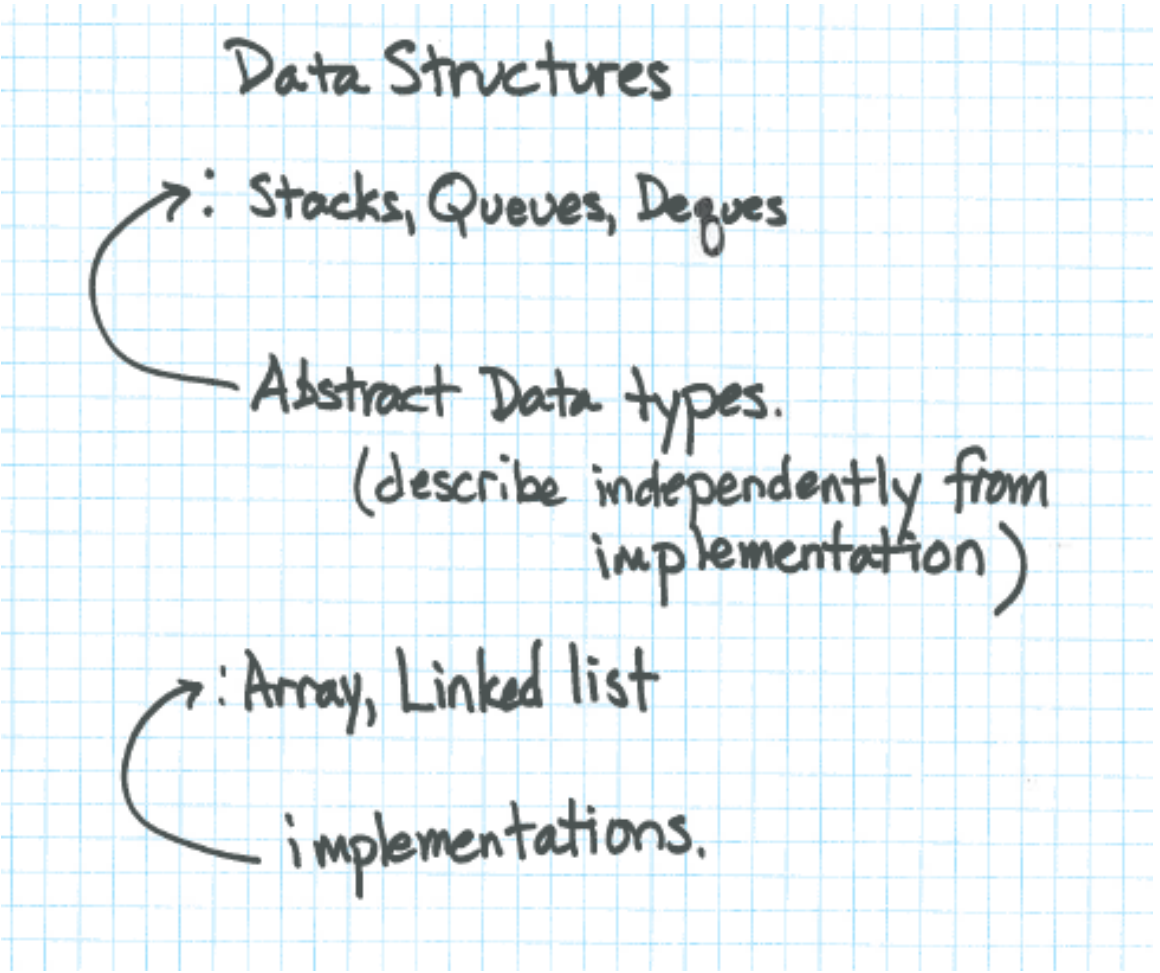
Meaning that ADT's can be implemented using either data types or data structures.

As for **data structures**:

> A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.

Many textbooks, use these words interchangeably. With more complex types, that can lead to confusion.

Basic Summary



Basic Summary (Shermer)

Study the following *Wikipedia* websites:

- Data Types
- Abstract Data Types
- Data Structures

Homework

Presentation Terminated