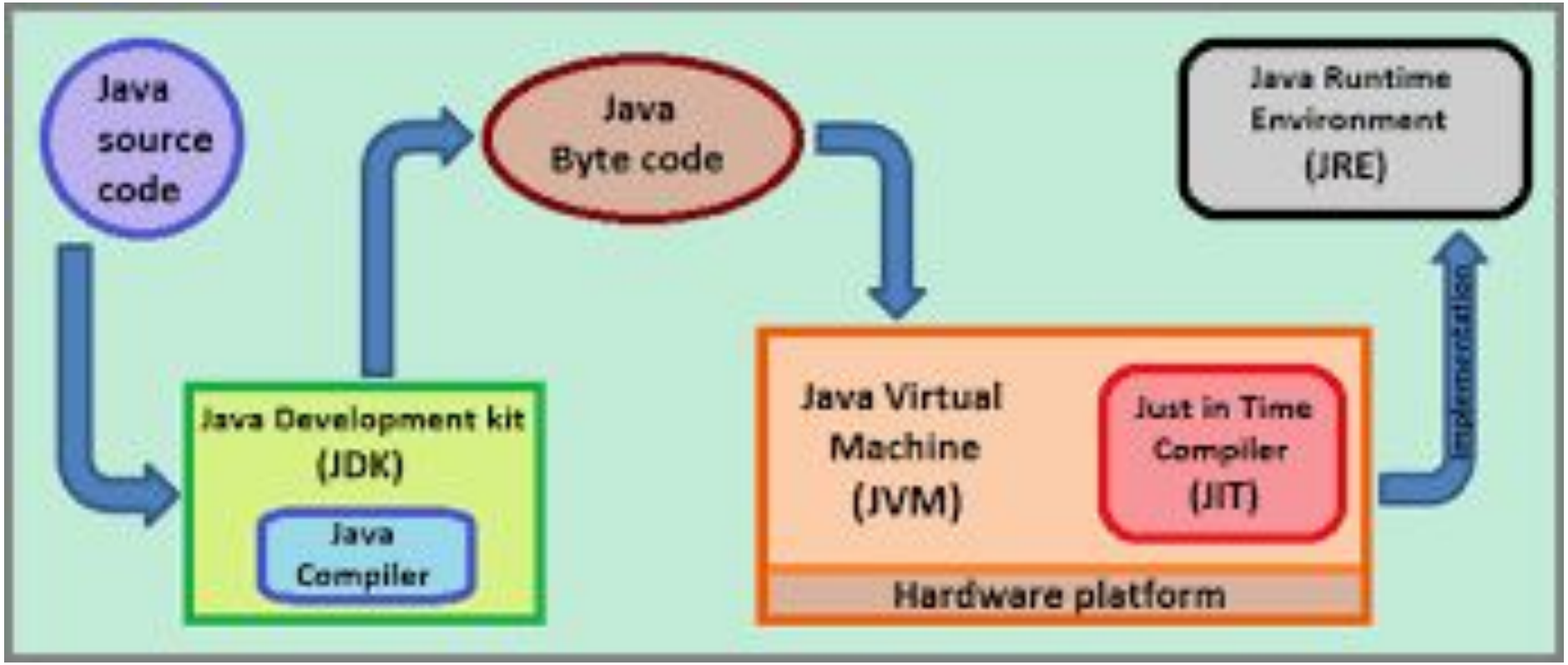
Java

WHY Java ?

- 1. Platform or architecture independent (Write once run anywhere!)
- 2. Simple & robust
- 3. Secure
- 4. Automatic memory management.
- 5. Inherent Multi threaded support
- 6. Object Oriented support -- Encapsulation, Inheritance & polymorphism , abstraction
- 7. Excellent I/O support
- 8. Inherent networking support for TCP/IP , UDP/IP programming & for URLs
- 9. Adds functional programming support.

Development & Execution of java application.



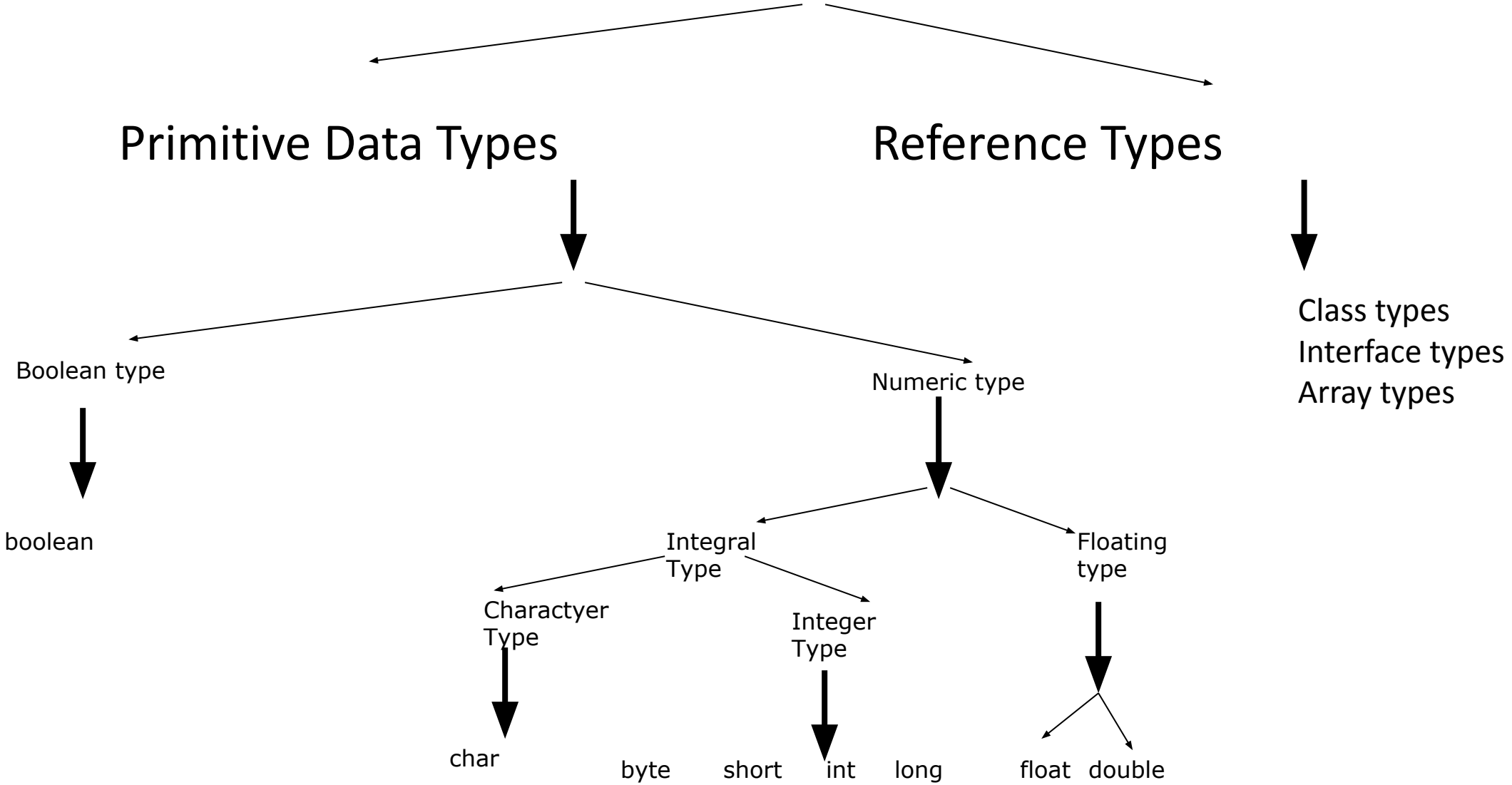
Language Basics

- Keywords
- Variables
- Conditional Statements
- Loops
- Data Types
- Operators
- Coding Conventions

Java Keywords

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
true	false	null			

Data types In Java



Data Types

- Integral Type
  - byte 8 bits
  - short 16 bits
  - int 32 bits
  - long 64 bits

- Textual Type
  - char 16 bits, UNICODE Character
  - String

Data Types

- Floating Point Type
  - float 32 bits
  - double 64 bits
- Boolean Type 1 bit
  - true
  - false

Variables

- A **variable** is a name given memory location. That memory is associated to a data type and can be assigned a value.

- int n;
- float f1;
- char ch;
- double d;

Variables conti…

- Assigning a value to a variable

- Initialization of a variable with a primary value

```
1. int  n1;
2. n1 =21 ;          // assignment
3. int   i2 = 18;     // initialization
4. char  ch = 'S';    // initialization
5. double d = 21.8;   // initialization
6. d = n1;           // assignment
7. float f1 = 16.13F;
```

Naming conventions

- class , interfaces , enum names- 1st letter of 1st word must start with upper case & then follow camel case notation.

  eg : HelloWorld

- data members/methods(functions) -- 1st must start with lower case & then follow camel case notation

  eg : performanceIndex

  public double calculateSalary(....) {...}

- constants -- all uppercase.

  eg : PI

Rules on Identifiers

- 1. Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_), cannot start with a number!
- 2. Can't use a Java keyword as an identifier.
- 3. Are Case sensitive

Access specifiers

- private
- default(package private) --no access modifier
- protected
- public

- Legal class level access specifiers -
- 1. default(scope=current package only)
- 2. public (scope=accessible from any where)

Basic rules

1. Java compiler doesn't allow accessing of un initialized data members.
2. Files with no public classes(default scoped) can have a name that does not match with any of the classes in the file .
3. A file can have more than one non public class.
4. There can be only one public class per source code file.
5. If there is a public class in a file, the name of the file must match the name

of the public class. For example, a class declared as public class Example {....}

must be in a source code file named Example.java.

6. Java compiler doesn't allow accessing of un-inited vars.

eg : int n;

sop(n);//error

Conversions regarding primitive types

Automatic conversions(widening) ---Automatic promotions

byte--->short--->int---> long--->float--->double

char ---> int

eg : char ch='a';

long --->float ---is considered automatic type of conversion(since float data type can hold larger range of values than long data type)

Rules ---

- src & dest - must be compatible, typically dest data type must be able to store larger magnitude of values than that of src data type.

- 1. Any arithmetic operation involving byte,short --- automatically promoted to --int
- 2. int & long ---> long
- 3. long & float ---> float
- 4. byte,short.....& float & double----> double

Narrowing conversion --- forced conversion(type-casting)

eg ---

double ---> int

float --> long

double ---> float

Basic Java Syntax
Operators
Control Statements

Primitive Types and Variables

- boolean, char, byte, short, int, long, float, double etc.
- These basic (or primitive) types are the only types that are not objects (due to performance issues).
- This means that you don’t use the new operator to create a primitive variable.
- Declaring primitive variables:

```
float initVal;
int retVal, index = 2;
double gamma = 1.2, brightness;
boolean valueOk = false;
```

Declarations

```
int data= 1.2;              // compiler error
boolean flag= 1;            // compiler error
double fract= 5 / 4;   // no error!
float ratio = 5.8f;         // correct
double fracto= 5.0 / 4.0;    // correct
```

- 1.2f is a float value accurate to 7 decimal places.
- 1.2 is a double value accurate to 15 decimal places.

Assignment

- All Java assignments are right associative

```
int a = 1, b = 2, c = 5;
a = b = c;
System.out.print(“a= “ + a + “b= “ + b + “c= “ + c);
```

- What is the value of a, b & c
- Done right to left: a = (b = c);

Basic Mathematical Operators

- `*  /  %  +  –` are the mathematical operators
- `*  /  %` have a higher precedence than `+` or `–`

```
double myVal = a + b % d – c * d / b;
```

- Is the same as:

```
double myVal = (a + (b % d)) – ((c * d) / b);
```

Relational Operators

==	Equal (careful)
!=	Not equal
>=	Greater than or equal
<=	Less than or equal
>	Greater than
<	Less than

Statements & Blocks

- A simple statement is a command terminated by a semi-colon:

```
name = “CDAC”;
```

- A block is a compound statement enclosed in curly brackets:

```
{
        name1 = “DAC”; name2 = “Java”;
}
```

- Blocks may contain other blocks

Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements:

Conditional Stmt: if, if else, switch

Looping: for, while, do while

Escapes: break, continue, return

if Statement – different syntax options

```
if
(expression)
   statement;
```

⇨ A single statement.

```
if
(expression)
   {
    statements;
   }
```

⇨ A block of statements.

```
if
(expression)
   statement;
else
   statement;
```

⇨ Single statement in the if and a single statement in the else.

```
if
(expression)
      statement;
else
{
   statements;
}
```

⇨ A single statement in the if and a block of statements in the else.

Conditional Operator

- The operator “ ? : ” is the only operator that takes three operands, each of which is an expression.
- The value of the whole expression equals the value of expr2 if expr1 is true, or equals the value of expr3 if expr1 is false.
- Syntax:

```
expr1 ? expr2 : expr3
```

switch Statement

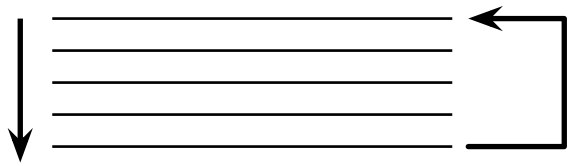
- The nested if can become complicated and unreadable.
- The switch statement is an alternative to the nested if.
- Syntax:

```
switch(expression)
{
    case constant expr:
            statement(s);
            [break;]
    case constant expr:
            statement(s);
            [break;]
    case constant expr:
            statement(s);
            [break;]
    default :
            statement(s);
            [break;]
}
```

- Usually, but not always, the last statement of a case is break.
- default case is optional.

Loops

- Loops break the serial execution of the program.
- A group of statements is executed a number of times.



There are three kinds of loops :

- **`while`**
- **`for`**
- **`do … while`**

While – Loop

- Syntax:

```
while (expression)
   Statement;
```

or

```
while (expression)
{Statements;}
```

- The loop continues to iterate as long as the value of expression is true (expression differs from zero).

- Expression is evaluated each time before the loop body is executed.

- The braces { } are used to group declarations and statements together into a compound statement or block, so they are syntactically equivalent to a single statement.

for - Loop

- Syntax:

```
for (expr1 ; expr2 ; expr3)
     statement;
```

or

```
for (expr1 ; expr2 ; expr3)
{
    statements;
}
```

- Is equivalent to:

```
expr1;
while (expr2)
{
   {statements;}
   expr3;
}
```

do while Loop

- Syntax:

```
do
{
   Statements;
}while (expression);
```

- The condition expression for looping is evaluated only after the loop body had executed.

break Statement

- We have seen how to use the break statement within the switch statement.

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

continue Statement

- In some situations, you might want to skip to the next iteration of a loop without finishing the current iteration.

- The **`continue`** statement allows you to do that.

- When encountered, **`continue`** skips over the remaining statements of the loop, but **`continues`** to the next iteration of the loop.

What is Scanner ?

- A class (java.util.Scanner) that represents text based parser(has inherent small ~ 1K buffer)
- It can parse text data from any source --Console input,Text file , socket, string

e.g. Scanner input = new Scanner(System.in);

```
System.out.print("Enter your name: ");
String name = input.next ();
System.out.println(“Your name is " + name);
input.close();
```