AP CSA

zhang si 张思

zhangsi@rdfz.cn

ICC 609

EXAM DESCRIPTION

Units	Exam Weighting
Unit 1: Primitive Types	**2.5–5%**
Unit 2: Using Objects	**5–7.5%**
Unit 3: Boolean Expressions and `if` Statements	**15–17.5%**
Unit 4: Iteration	**17.5–22.5%**
Unit 5: Writing Classes	**5–7.5%**
Unit 6: Array	**10–15%**
Unit 7: `ArrayList`	**2.5–7.5%**
Unit 8: 2D Array	**7.5–10%**
Unit 9: Inheritance	**5–10%**
Unit 10: Recursion	**5–7.5%**

1 Java Language Features

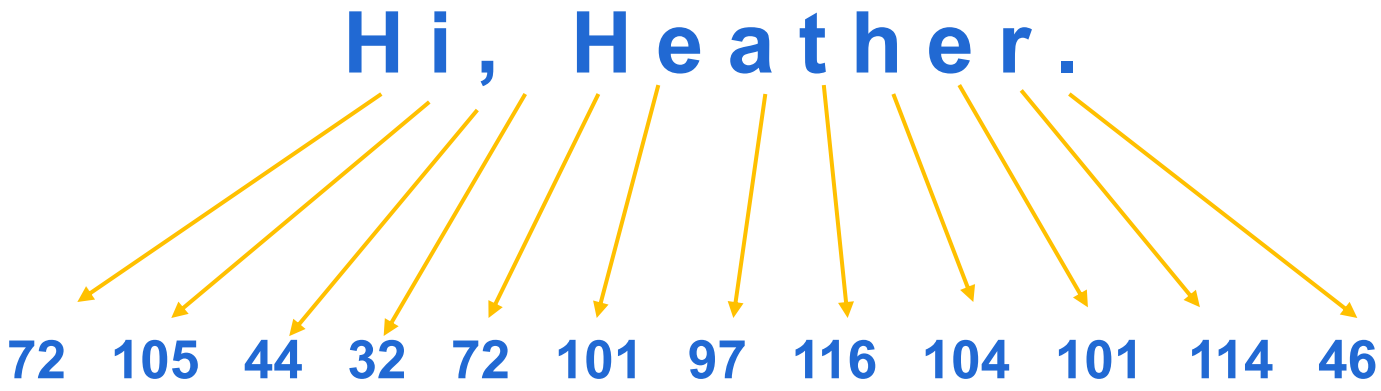
- Binary & Decimal &Octal & Hexadecimal number System
  - conversions
- Data type & Variables
  - int & double
  - String literal
  - variables & indetifiers
- Operators & Expressions
  - Arithmetic operator
  - Assignment operator
  - casting
- Java Program Structure & the print method
  - helloworld.java
  - System.out.print() &System.out.println()
  - errors & exceptions

Digital Information

- Computers store all information digitally:
  - numbers
  - text
  - graphics and images
  - video
  - audio
  - program instructions
- In some way, all information is *digitized* - broken down into pieces and represented as numbers

Representing Text Digitally

- For example, every character is stored as a number, including spaces, digits, and punctuation
- Corresponding upper and lower case letters are separate characters



Binary Numbers

- Once information is digitized, it is represented and stored in memory using the binary number system
- A single binary digit (0 or 1) is called a bit
- Devices that store and move information are cheaper and more reliable if they have to represent only two states
- A single bit can represent two possible states, like a light bulb that is either on (1) or off (0)
- Permutations of bits are used to store values

Bit Permutations

1 bit	2 bits	3 bits	4 bits	
0	00	000	0000	1000
1	01	001	0001	1001
	10	010	0010	1010
	11	011	0011	1011
		100	0100	1100
		101	0101	1101
		110	0110	1110
		111	0111	1111

Each additional bit doubles the number of possible permutations

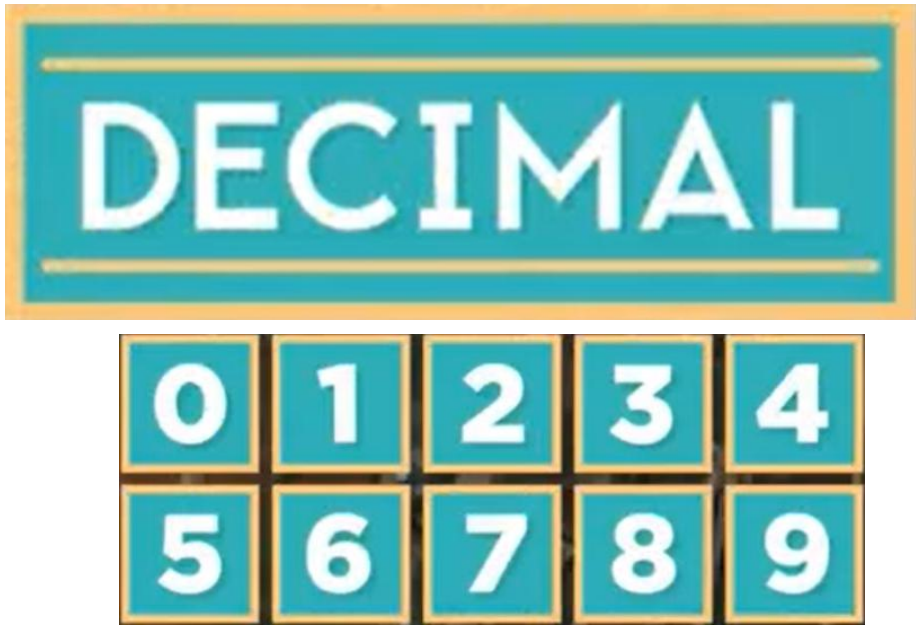
Bit Permutations

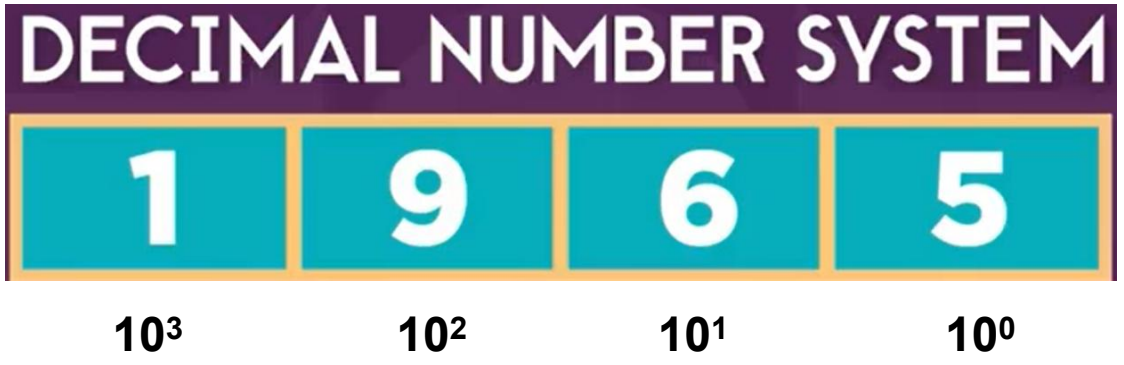
- Each permutation can represent a particular item
- There are 2^N permutations of N bits
- Therefore, N bits are needed to represent 2^N unique items

How many items can be represented by

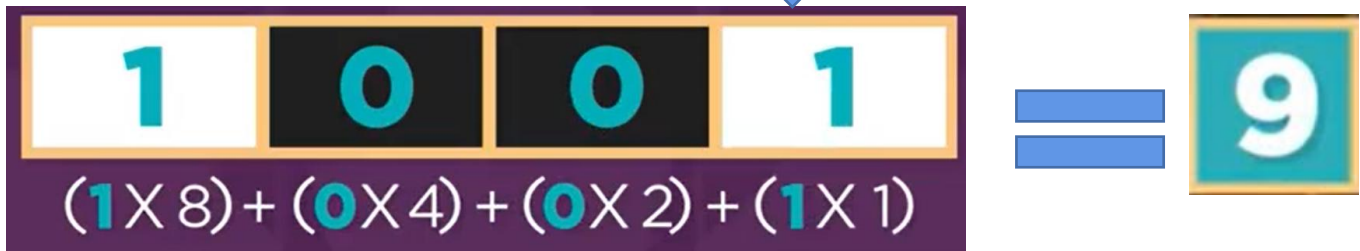
- **1 bit ?** — $2^1 = 2$ items
- **2 bits ?** — $2^2 = 4$ items
- **3 bits ?** — $2^3 = 8$ items
- **4 bits ?** — $2^4 = 16$ items
- **5 bits ?** — $2^5 = 32$ items

Binary & Decimal





Binary to Decimal



Binary & Decimal

- Decimal to Binary——Successive Division

$$(X_3\ X_2\ X_1\ X_0)_b = X_3\ b^3 + X_2\ b^2 + X_1\ b^1 + X_0\ b^0$$

$$(1\ 0\ 1\ 0)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (10)_{10}$$

Assume:

$$(X_3\ X_2\ X_1\ X_0)_b = X_3\ b^3 + X_2\ b^2 + X_1\ b^1 + X_0\ b^0 = N$$

$$\frac{N}{b} = \underbrace{X_3\ b^2 + X_2\ b^1 + X_1\ b^0}_{N_1} + X_0$$

$$\frac{N_1}{b} = \underbrace{X_3\ b^1 + X_2\ b^0}_{N_2} + X_1$$

$$\frac{N_2}{b} = \underbrace{X_3\ b^0}_{N_3} + X_2$$

$$\frac{N_3}{b} = 0 + X_3$$

$(27)_{10} = (11011)_2$

		Remainder
2	27	1
2	13	1
2	6	0
2	3	1
2	1	1
	0	

Octal & Hexadecimal

- Hexadecimal number system: base is 16
  - symbols 0 – 9 and A – F (occasionally a – f), where A represents 10, and F represents 15.
  - in Java, the prefix "0x" or "0X" is used: 0xC2A.
  - On the AP exam, the representation is likely to be with the subscript hex: $C2A_{hex}$
- Octal number system:base is 8
  - symbols 0 – 7
  - On the AP exam, the representation is likely to be with the subscript hex: 135_{oct}

Try to transform $(348)_{dec}$ into octal number and hexadecimal number:

$(348)_{dec}=(534)_{cot}$

$(534)_{oct}=5*8^2+3*8^1+4*8^0=320+24+4=348$

		Reminder
8	348	
8	43	4
8	5	3
	0	5

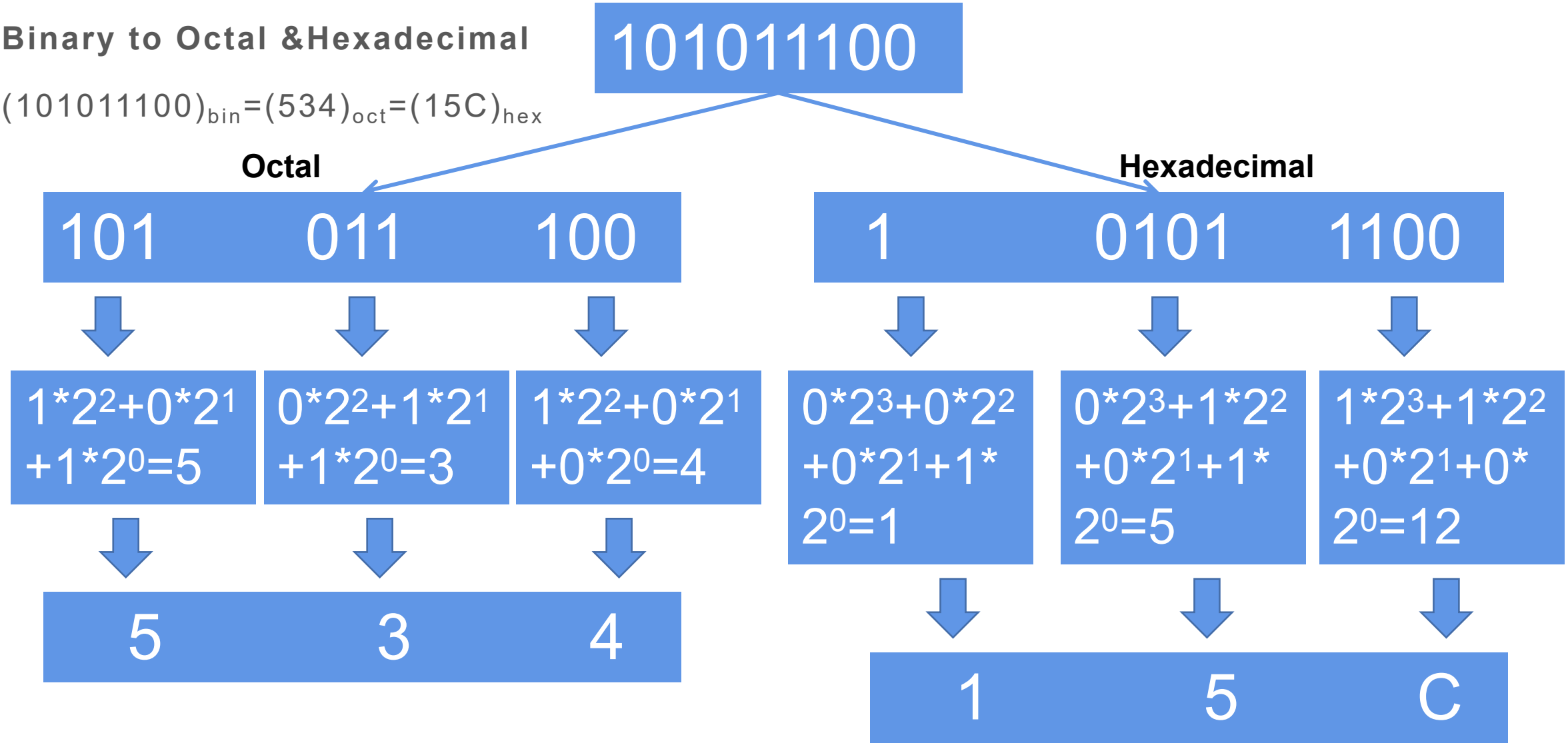
$(348)_{dec}=(15C)_{hex}$

$(14C)_{hex}=1*16^2+5*16^1+12*16^0=256+80+12=348$

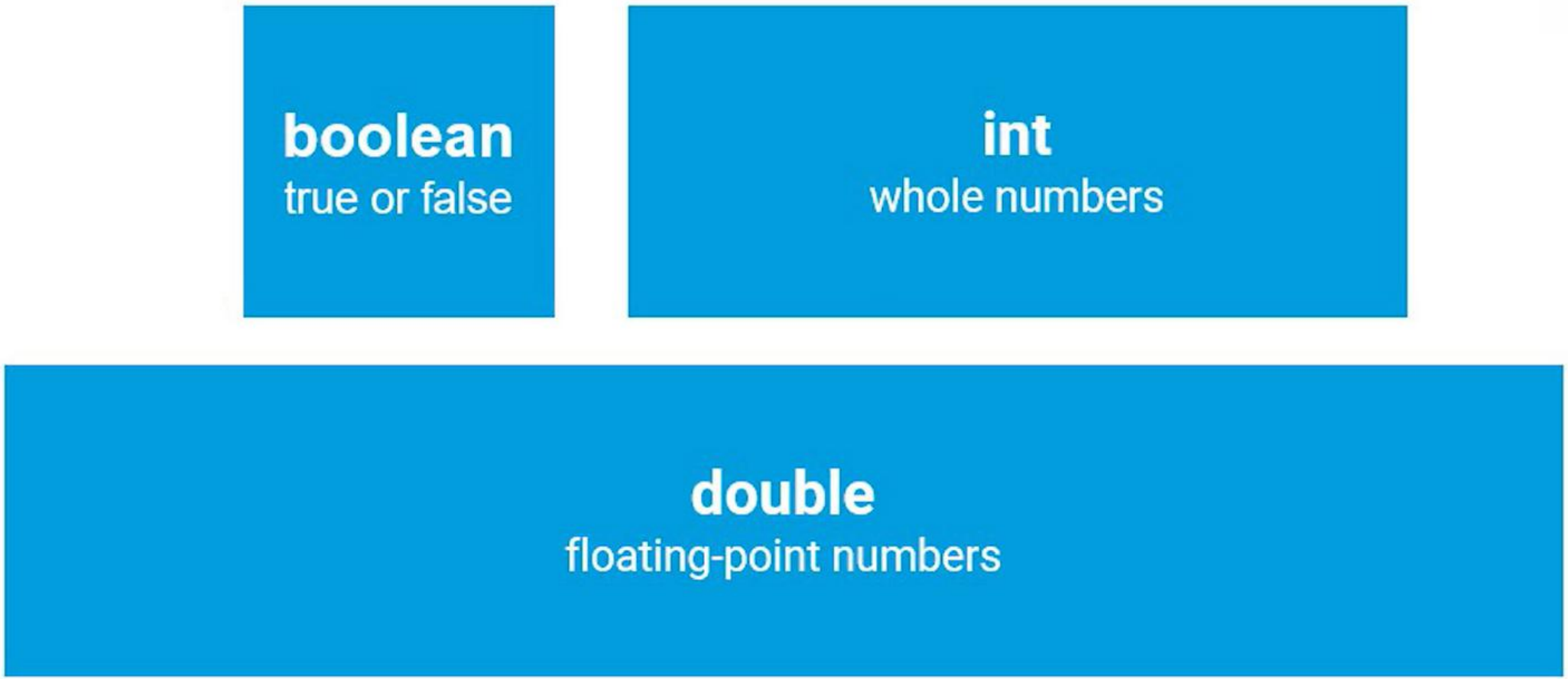
		Reminder
16	348	
16	21	12
16	1	5
	0	1

Octal & Hexadecimal

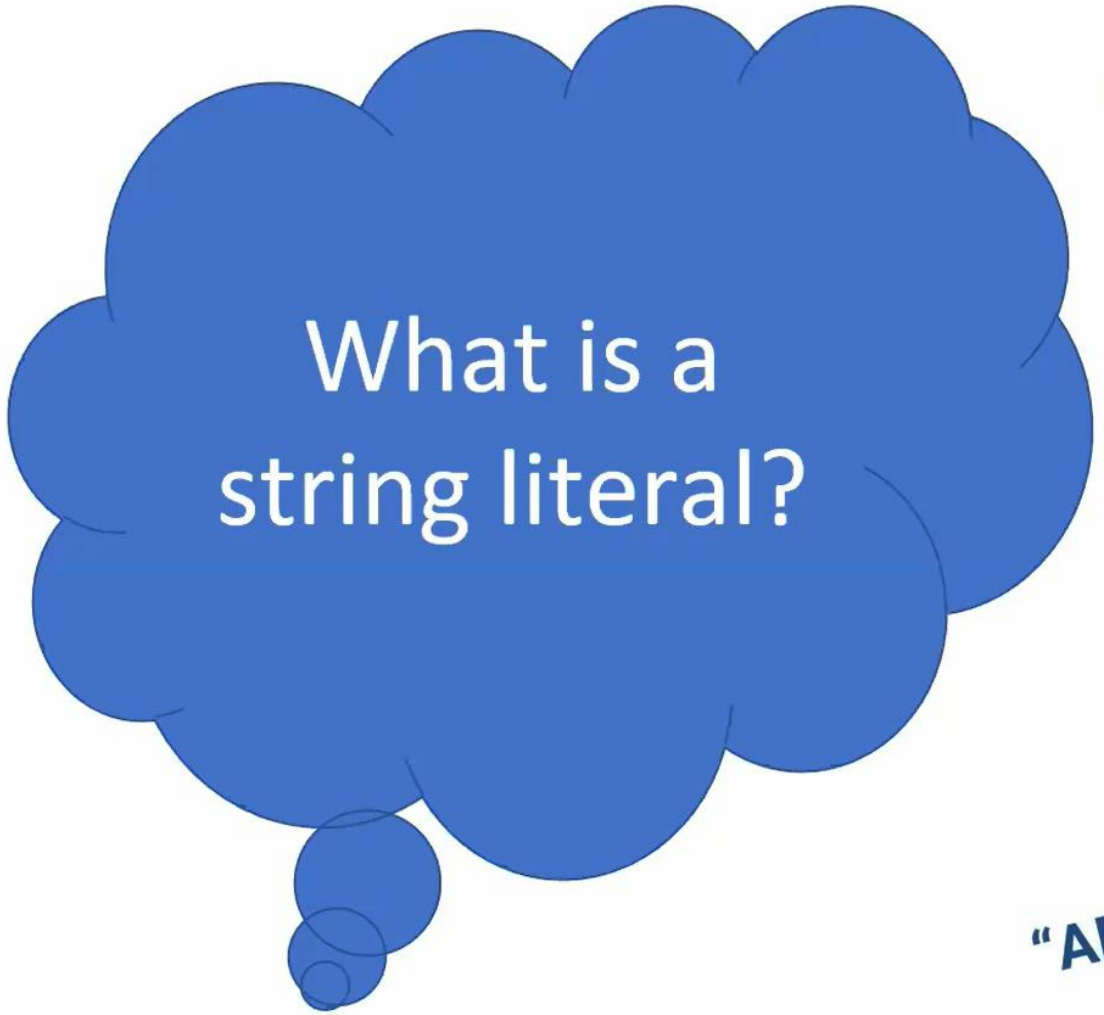
- **Binary to Octal &Hexadecimal**
- $(101011100)_{bin}=(534)_{oct}=(15C)_{hex}$



Primitive Data



String literal



"ABC123#$%abc"

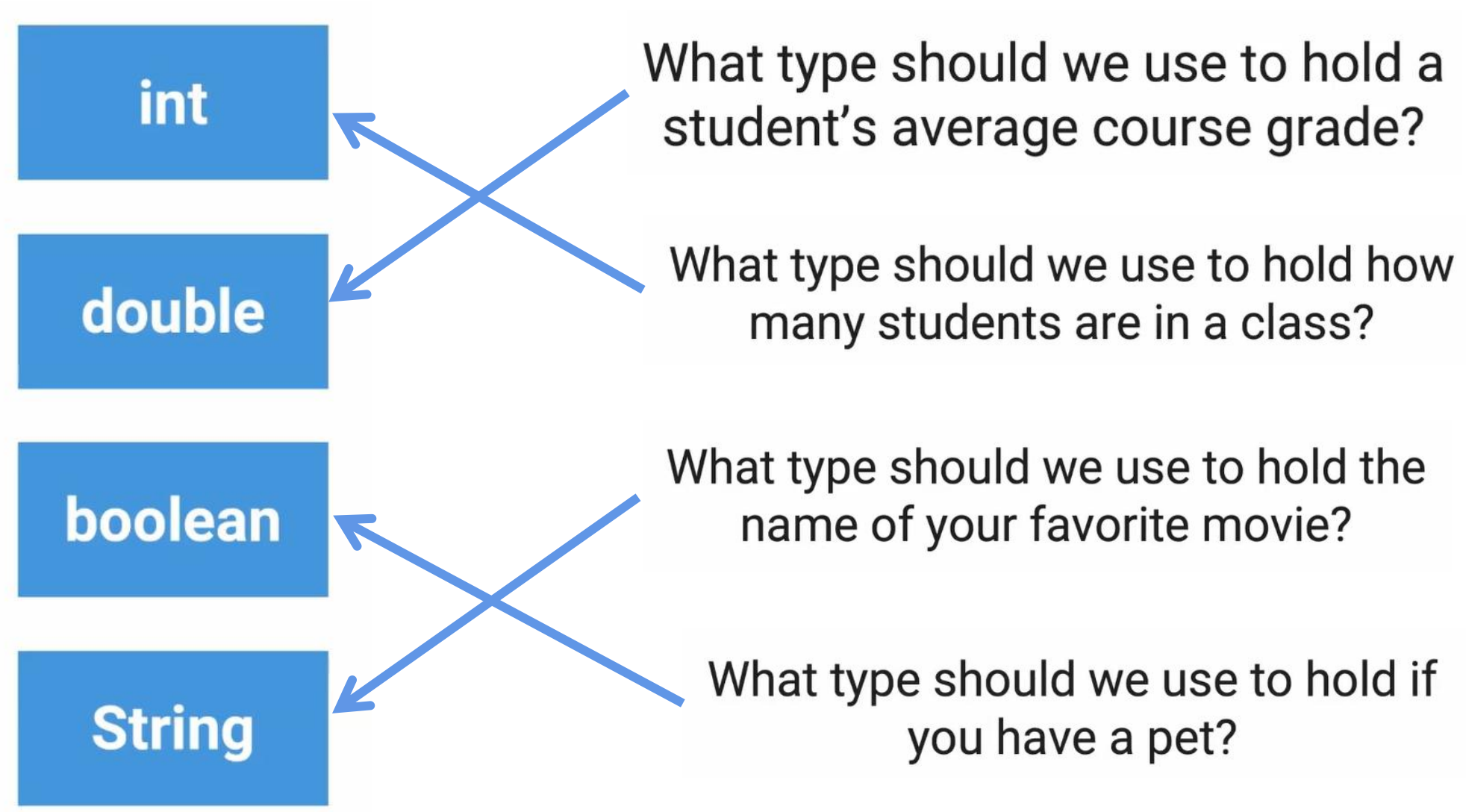
"Have a GREAT day!!"

A string literal in java is an exact sequence of characters (letters, numbers, or symbols) which are enclosed between two quotation marks.

"AP CS A Rocks!!!"

"This is a string literal."

choose the appropriate data type



Range of integer & floating-point number

- Data types are limited in how much information they can store!

int:
4Byte =32 Bit
range : -2^{31}~2^{31}-1

int
2147483647 (`Integer.MAX_VALUE`)
-2147483648 (`Integer.MIN_VALUE`)

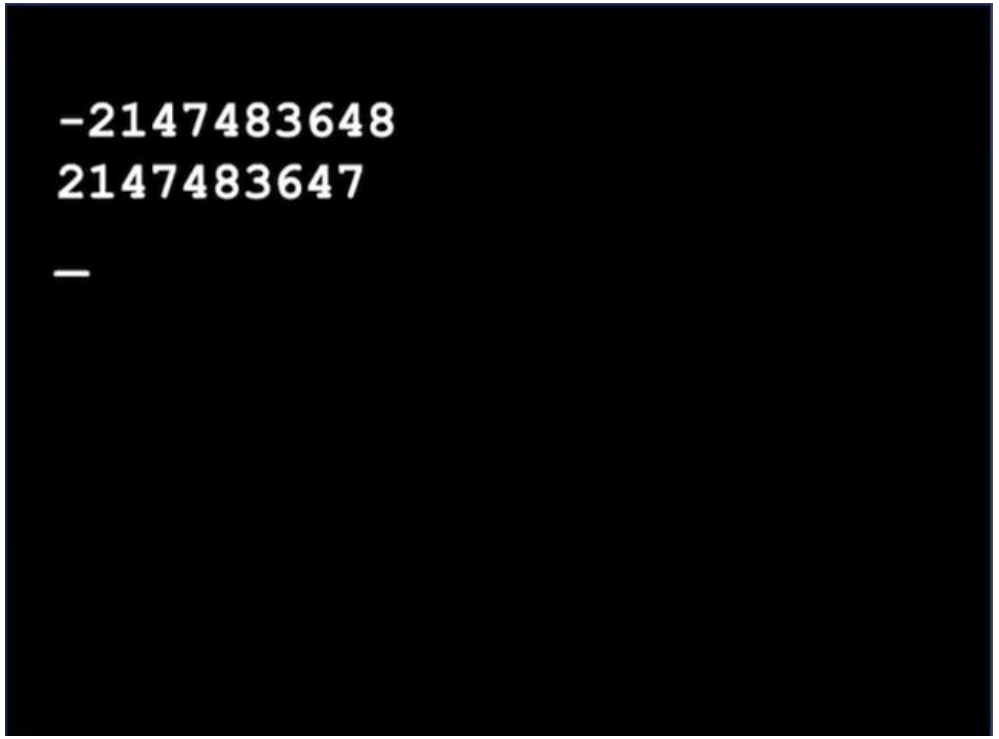
double:
8Byte =64 Bit
1Bit for sign (+ or -)
11Bits for exponent
52 Bits for Significand precision

double
up to 14-15 digits

Range of integer

```
public class TooBigNumber {

    public static void main(String[] args) {

        int posInt = Integer.MAX_VALUE;
        int negInt = Integer.MIN_VALUE;
        System.out.println(posInt + 1);
        System.out.println(negInt - 1);

    }

}
```

- **The code will compile** because there's nothing syntactically wrong with this statement,and the program **will run**
- but the number that's going to **output is not what we expected** because it is **outside** of the allowable range for an integer



Data-type:Double

- round-off error

When floating-point numbers are converted to binary, most cannot be represented exactly, leading to round-off error. These errors are compounded by arithmetic operations.

```
public class Test{
    public static void main(String args[]){
        System.out.println(0.05+0.01);
        System.out.println(1.0-0.42);
        System.out.println(4.015*100);
        System.out.println(123.3/100);
    }
}
```

Data-type:Double

- compare two double-type: **Do not use ==**

```
public class Test{
    public static void main(String args[]){
        double A=0.1+0.1+0.1+0.1+0.1+0.1;
        double B=0.1*6;
        if (A==B){System.out.println("True");}
            else {System.out.println("False");}

        if (Math.abs(A-B)<1e-3){System.out.println("True");}
            else {System.out.println("False");}
    }
}
```

False

True

Extended Reading

Optional topic

Comparing Floating-Point Numbers

Because of round-off errors in floating-point numbers, you can't rely on using the `==` or `!=` operators to compare two `double` values for equality. They may differ in their last significant digit or two because of round-off error. Instead, you should test that the magnitude of the difference between the numbers is less than some number about the size of the machine precision. The machine precision is usually denoted ϵ and is typically about 10^{-16} for double precision (i.e., about 16 decimal digits). So you would like to test something like $|x - y| \leq \epsilon$. But this is no good if x and y are very large. For example, suppose $x = 1234567890.123456$ and $y = 1234567890.123457$. These numbers are essentially equal to machine precision, since they differ only in the 16th significant digit. But $|x - y| = 10^{-6}$, not 10^{-16}. So in general you should check the *relative* difference:

$$\frac{|x - y|}{\max(|x|, |y|)} \leq \epsilon$$

To avoid problems with dividing by zero, code this as

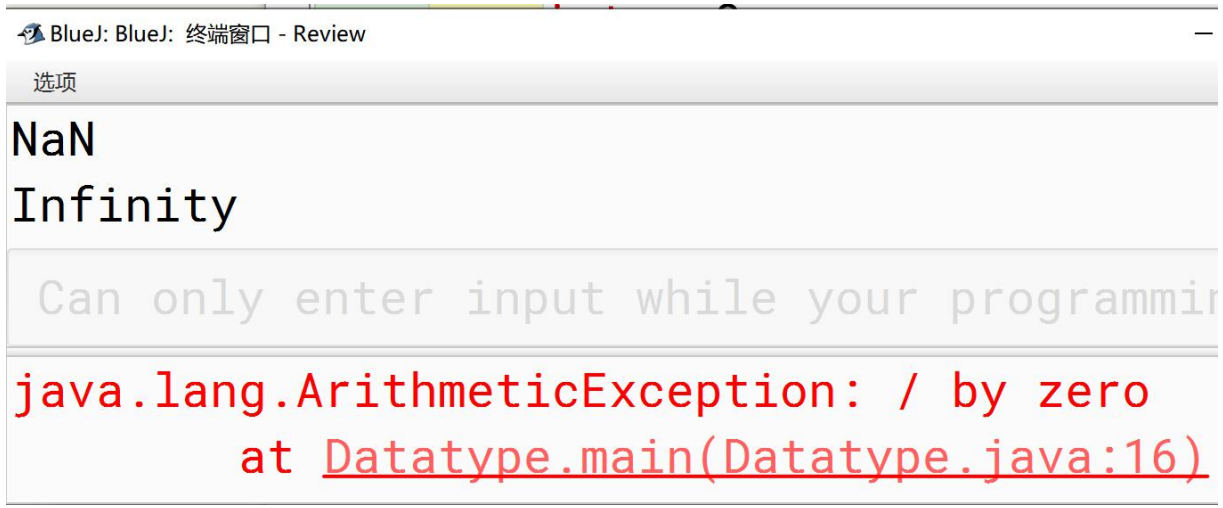
$$|x - y| \leq \epsilon \max(|x|, |y|)$$

Data-type:Double

```
public class Datatype
{
    public static void main(String[] args){
        double a=0;
        double b=8;
        int c=8;
        System.out.println(a/0);
        System.out.println(b/0);
        System.out.println(c/0);
    }
}
```

编译完成 - 没有语法错误

```java
public class Datatype
{
    public static void main(String[] args){
        double a=0;
        double b=8;
        int c=8;
        System.out.println(a/0);
        System.out.println(b/0);
        System.out.println(c/0);
    }
}
```



In Java, no exceptions are thrown for floating-point operations. There are two situations you should be aware of:

- When an operation is performed that gives an undefined result, Java expresses this result as `NaN`, "not a number." Examples of operations that produce `NaN` are: taking the square root of a negative number, and `0.0` divided by `0.0`.
- An operation that gives an infinitely large or infinitely small number, like division by zero, produces a result of `Infinity` or `-Infinity` in Java.

Variables

consider we need to writedown some information for different students:

Mary has 2 notebooks for computer science class.

Spencer has 4 notebooks for English class.

Sofia has 6 notebooks for history class.

What if we wanted to write this sentence for everyone in class?

What about everyone in the school?

How could we do this more quickly?

Variables

Mary has **2** notebooks for **computer science** class.

Spencer has **4** notebooks for **English** class.

Sofia has **6** notebooks for **history** class.

firstName has **numNotebooks** notebooks for **course** class.

Variables

Variables

Remeber:
When a variable is referenced in a program, its current value is used

VARIABLE

a name given to a memory location that is holding a value

```
String name = "Mark Dean";
```

Declaring and initializing a String variable

```
double avgGrade;
```

Declaring a double variable

```
avgGrade = 92.6;
```

initializing the `avgGrade` variable

```
final int NUMCLASSES = 4;
```

Defining a constant

```
boolean isEnrolled = true;
```

Declaring and initializing a boolean variable

Identifiers

- *Identifiers are the words a programmer uses in a program*
- An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign
- Identifiers cannot begin with a digit
- Java is *case sensitive* - `Total`, `total`, and `TOTAL` are different identifiers
- By convention, Java programmers use different case styles for different types of identifiers, such as
  - *title case for class names* - `Lincoln`
  - *upper case for constants (final)* - `MAXIMUM`

Identifiers

- Sometimes we choose identifiers ourselves when writing a program (such as `Lincoln`)
- Sometimes we are using another programmer's code, so we use the identifiers that they chose (such as `println`)
- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

Reserved Words

- The Java reserved words:

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

https://www.javatpoint.com/java-keywords

Operators

- An *expression* is a combination of one or more operands and their operators
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Arithmetic Operators

Operator	Meaning	Example
+	addition	`3 + x`
-	subtraction	`p - q`
*	multiplication	`6 * i`
/	division	`10 / 4  //returns 2, not 2.5!`
%	mod (remainder)	`11 % 8  //returns 3`

- An *expression* is a combination of one or more operands and their operators
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Operation	Result
int + int	int
int – int	int
int * int	int
int / int	int
int % int	int

```
System.out.println(17+5);
System.out.println(17.0+5);
System.out.println(17/5);
System.out.println(17.0/5);
```

Result:

```
22
22.0
3
3.4
```

Operation	Result
double + double	double
double – double	double
double * double	double
double / double	double
double % double	double

Operation	Result
double + int	double
double – int	double
double * int	double
double / int	double
double % int	double

Operators

Compound Assignment Operator	... same as...
`x += 7;`	`x = x + 7;`
`x -= 3;`	`x = x - 3;`
`x *= 10;`	`x = x * 10;`
`x /= 5;`	`x = x / 5;`
`x %= 3;`	`x = x % 3;`

Increment/Decrement Operator has the **higher** Precedence than * / %

```
int d=3;
System.out.println(17/d++);
System.out.println(d);
```

result is :

```
5
4
```

Increment/Decrement Operator	... same as...	... same as...
`x++;`	`x = x + 1;`	`x += 1;`
`x--;`	`x = x - 1;`	`x -= 1;`

Operator Precedence

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4       1   3     2
```

Then the result is stored in the variable on the left hand side

Operator	Type	Direction
( )	Parenthesis	Left to Right
* / %	Multiplication Division Modulus	Left to Right
+ -	Addition Subtraction	Left to Right
Other operators that will be discussed later		
=	Assignment Operator	Right to Left

Operator Precedence

3 + 4 * (int)7.3 * (8 – 6) % 5 / 2

3 + 4 * (int)7.3 * **2** % 5 / 2

3 + 4 * **7** * 2 % 5 / 2

3 + **28** * 2 % 5 / 2

3 + **56** % 5 / 2

3 + **1** / 2

3 + **0**

3 + 0

3

( )	parentheses
++ --	increment & decrement
( )	casting
* / %	multiplicative operations
+ -	additive operations & string concatenation
= *= /= %= == -=	assignment operators

* *not a complete table*

Data Conversions

- In Java, data conversions can occur in three ways:
  - **assignment conversion**
  - **arithmetic promotion**
  - **casting**
- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
  - Only widening conversions can happen via assignment
  - eg: **double a=5.35;**
  - **a=10;**
  - *Arithmetic promotion* happens automatically when operators in expressions convert their operands
    - eg: **int a=5; double b=3.5, the type of (b+a) is double**

Casting

- We use casting in Java to change the data type of a variable from one type to another

examples:

```
(int) (2.5 * 3.0)      7
(double) 25 / 4        6.25
6 / (double) 5         1.2
(int) 12 / 5           2
```

```
(int) 3.0 / 4      →   0
(double) 3 / 4     →   0.75
(double) (3 / 4)   →   0.0
```

round off

```
(int) (6.75 + 0.5)     7
(int) (12.59 + 0.5)    13
(int) (-8.63 - 0.5)    -9
(int) (-10.8 - 0.5)    -11
```

Casting has the **higher** Precedence than * / %

Java Program Structure

- In the Java programming language:
  - A program is made up of one or more *classes*
  - A class contains one or more *methods*
  - A method contains program *statements*

- These terms will be explored in detail throughout the course

- A Java application always contains a method called `main`

Hello world

```
/*
 * Programmer: Mr. Schultz
 * Date: 7/23/2020
 * Purpose: Demonstrate the System class methods
 */

public class HelloWorld {

    public static void main(String[] args) {

        System.out.print("AP ");
        System.out.print("CS ");
        System.out.print("A ");
        System.out.println("Rocks!");
        System.out.println("Hello World");

    }
}
```

Block Comments
the compiler will ignore any text between /*and*/

Class Declaration
Indentifies the name,the start,and the end of the class.
The class name Must match the file name.(i.e. HelloWorld.java)

main Method
Controls all of the action in the program

System.out
object that generates output to the console

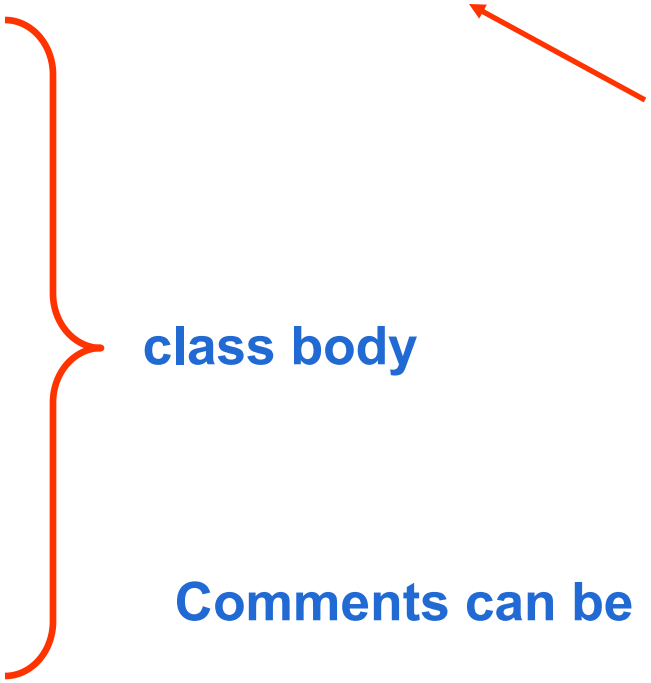
Comments

- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment
* this is a javadoc comment   */
```

Java Program Structure

```
//  comments about the class
public class HelloWorld
{


}
```



Java Program Structure

```
//  comments about the class
public class HelloWorld
{

   //  comments about the method
   public static void main (String[] args)
   {

   }

}
```

the Print method

- **`System.out.print()`**
  - display whatever in `()` to the screen
- **`System.out.println()`**
  - display whatever in `()` to the screen, and move the cursor **to the next line**
- String literal: is an exact sequence of characters(letters,numbers,or symbols)which are enclosed between two quotation marks.
  - A string literal cannot be broken across two lines in a program
- The *string concatenation operator* (+) is used to append one string to the end of another
  - eg:`System.out.println(”hahaha”+”xixixi”);`
  - eg:`System.out.println(5+”hahaha”);`
  - eg:`System.out.println(5+3);`

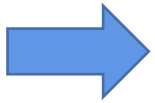
the plus + operator performs depends on the type of the information on which it operates

Escape Sequences

- What if we wanted to print a double quote character?

Escape Sequence	Meaning
\b	backspace
\t	tab
\n	newline
\r	carriage return
\"	double quote
\'	single quote
\\	backslash

```
eg:System.out.println ("I said \"Hello\" to you.");
```



```
I said "Hello" to you.
```

Syntax and Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program

- The *semantics* of a program statement define what that statement means (its purpose or role in a program)

- A program that is syntactically correct is not necessarily logically (semantically) correct

- A program will always do what we tell it to do, not what we meant to tell it to do

Errors

- A program can have three types of errors
- The compiler will find syntax errors and other basic problems (***compile-time errors***)
    - If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (***run-time errors***)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (***logical errors***)

Errors

- An attempt to divide an integer by zero will result in an Arithmetic Exception to occur.

Type	Example	Detection

Syntax/Compiler Error	`System.ot.print("Hi");`
	`system.out.print("Hi");`
	`System.out.print("Hi")`
		While some IDEs will detect syntax errors as code is typed, syntax errors are identified when the program is compiled. A program will not compile or run while syntax errors are present.

Exception or run-time error	The program attempts to divide a number by 0.	Exceptions occur while the program is running and will cause the program to terminate abnormally. A program "throws an exception".
Logic Error	The programmer accidentally uses a minus (-) instead of plus (+) when finding the sum of two numbers.	Logic errors are usually detected after a program has been run when *actual* output is compared to *anticipated* output.

Exceptions

- An exception is an error condition that occurs during the execution of a Java program.
- An unchecked exception is one where you don't provide code to deal with the error. Such exceptions are automatically handled by Java's standard exception-handling methods, which terminate execution. You now need to fix your code!

The following exceptions are in the AP Java subset:

Exception	Examples
`ArithmeticException`	If you divide an integer by zero
`NullPointerException`	If you use an uninitialized object to call method
`ClassCastException`	If you cast a superclass to a wrong subclass
`ArrayIndexOutOfBoundsException`	If you use a negative array index
`IndexOutOfBoundsException`	if Index is negative or larger than the length of the string
`IllegalArgumentException`	a parameter does not satisfy a method's precondition