1

Programming Review Part 02

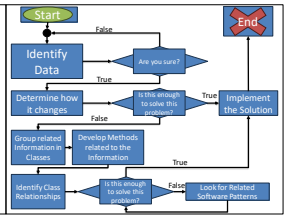
2

Procedural Programming

3

Problem Solving

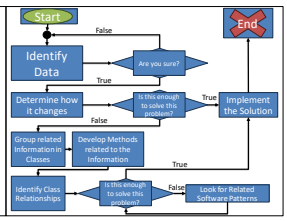
1. Identify your Data.
2. Determine how the data changes over time.
3. Consider structures for both behavior and data.
4. Group together (encapsulate) related information into Classes of Objects.
5. Develop functionality / methodologies that relates to the behavior of your Objects.
6. Further identify relationships between the Classes and optimize the structure.
7. Determine if there exists software patterns that may assist.



4

Problem Solving

1. Identify your data.
   - What information do we need to keep in our memory?
   - What information is **variable** and what information is **constant**?
   - How do we represent the data?
   - What is the type of data?
     - Numeric? Whole or Decimal?
     - Characters?
     - Words/Strings?
     - Collections? Other Structures?
   - Look for the **Nouns**



5

Variables

- Stores changeable information
- Containers for Data
- Declaring
  - Creates a Variable
  - (Data) Type
  - Identifier
- Spoken:
  - "Reserve space in memory for this type called this identifier" or
  - "Find some space to store this type of data"

Syntax

```
<<type>> <<identifier>>;
```

Example

```
double j;
```

6

Variables Types

- Type corresponds to bytes in memory
- Only use the type when declaring
- Programming Languages are either
  - –Strongly Typed
  - –Weakly Typed
- Primitive Types
- Object Types
  - –Reference *points to*
  - –Contents
- **Bold** are the most commonly used primitive data types in this course

Primitive Types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	**4 bytes**	**Stores whole numbers from -2,147,483,648 to 2,147,483,647**
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	**8 bytes**	**Stores fractional numbers. Sufficient for storing 15 decimal digits**
boolean	**1 bit**	**Stores true or false values**
char	**2 bytes**	**Stores a single character/letter or ASCII values**

7

Variables Types

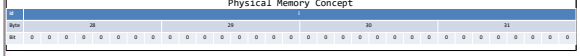
Example

```
int i;
double j;
char o;
```

Memory Concept

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	0.0	32
o	'\u0000'	40
Future Variable	-	42
…	…	…

Physical Memory Concept



8

Variable Identifiers

- Gives the variable a "name"
- Use the name to retrieve and store information
- Case sensitive
  - –int test, int TEST, int tEsT would be 3 different identifiers
- "Camel Casing" common practice used for identifiers
- Identifiers cannot
  - –Start with a Digit
  - –Have Spaces
  - –Match a reserved word
- Identifiers should avoid
  - –Special Characters
  - –Confusing names

Good Examples

```
int test01;
double largeValues;
boolean inClass;
```

Bad Examples

```
int 1Test;//Started with a digit
double big vals;//Used a space
boolean class;//Class is a reserved word
```

9

Assignment Operator

- The equals symbol "=" is the **Assignment Operator**
- Stores values found on the right hand side (RHS) of the operator into the identifier found on the left hand side (LHS)
- Assignments are valid if the type matches are is at least compatible
  - –Primitive types can be stored in other primitive types as long the type's byte amount is less than or equal to value being stored
  - –Otherwise "type casting" is required
  - –Type casting does not round it cuts off everything past the decimal point "."
- Spoken:
  - –"Store this value in this container"

Syntax

```
<<identifier>> = <<value>>;
```

Examples

```
i = 4;
j = 22.3;
o = 'h';
i = (int)j;//Type cast from double to int
//Value stored in "i" is 22
```

10

Assignment Operator

- Declare and assigning initial values
  - –Good programming practice
  - –Shortens two statements into one
  - –Types are not still used after the declaration

Example

```
int i = 4;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory Concept

Identifier	Contents	Byte Address
…	…	…
Future Variable	-	28
…	…	…

11

Assignment Operator

- Declare and assigning initial values
  - –Good programming practice
  - –Shortens two statements into one
  - –Types are not still used after the declaration

Example

```
int i = 4;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory Concept

Identifier	Contents	Byte Address
…	…	…
i	0	28
Future Variable	-	32
…	…	…

12

Assignment Operator

- Declare and assigning initial values
  - –Good programming practice
  - –Shortens two statements into one
  - –Types are not still used after the declaration

Example

```
int i = 4;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory Concept

Identifier	Contents	Byte Address
…	…	…
i	4	28
Future Variable	-	32
…	…	…

13

Assignment Operator

- Declare and assigning initial values
  - –Good programming practice
  - –Shortens two statements into one
  - –Types are not still used after the declaration

Example

```
int i = 4;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory Concept

Identifier	Contents	Byte Address
…	…	…
i	4	28
j	0.0	32
Future Variable	-	40
…	…	…

14

Assignment Operator

- Declare and assigning initial values
  - –Good programming practice
  - –Shortens two statements into one
  - –Types are not still used after the declaration

Example

```
int i = 4;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory Concept

Identifier	Contents	Byte Address
…	…	…
i	4	28
j	22.3	32
Future Variable	-	40
…	…	…

15

Assignment Operator

- Declare and assigning initial values
  - –Good programming practice
  - –Shortens two statements into one
  - –Types are not still used after the declaration

Example

```
int i = 4;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory Concept

Identifier	Contents	Byte Address
…	…	…
i	4	28
j	22.3	32
o	'h'	40
Future Variable	-	
…	…	…

16

Assignment Operator

- Declare and assigning initial values
  - –Good programming practice
  - –Shortens two statements into one
  - –Types are not still used after the declaration

Example

```
int i = 4;
double j = 22.3;
char o = 'h';
i = (int)j; //Typecast
```

Memory Concept

Identifier	Contents	Byte Address
…	…	…
i	22	28
j	22.3	32
o	'h'	40
Future Variable	-	
…	…	…

17

Constants

- Establishes a value that cannot change
- Great for avoiding "magic numbers"
- Good programming practice
  - –Make the scope public
  - –Make it static
  - –Capitalize all characters in the identifier

Syntax

```
public static final <<type>> <<identifier>> = <<value>>;
```

Examples

```
public static final double PI = 3.14159;
public static final int BOARD_SIZE = 10;
```

18

Math Operators

- Performs computation and then assigns the results
- Order of Operations
- Basic Math Operators
  - –Addition "+"
  - –Subtraction "-"
  - –Multiplication "*"
  - –Division "/"
- Mod Operator "%"
  - –Returns the remainder after division
  - –Ex: 15 % 2 = 1

Syntax

```
<<identifier>> = <<value>> <<operator>> <<value>>;
```

Examples

```
//Variables
int value = 64 % i + 32;
//Constants
public static final double PI = 3.14159;
public static final double PI_SQ = PI*PI;
```

19

Math Operators

- Compute and Assign (C&A) Operators
  - –Shorthand for applying some operator and value to a variable
  - –Same as:
    - <<identifier>> = <<identifier>> <<operator>><<value>>;
    - Ex: i = i+1; i+=1; i++; //Same statements
- Common Versions
  - –"++" – add and assign
  - –"-=" – subtract and assign
  - –"*=" – multiply and assign
  - –"/=" – divide and assign
  - –"%=" – mod and assign
- Special versions
  - –"++" – Increase by 1
    - Same as "+= 1"
  - –"--" – Decrease by 1
    - Same as "-=1"

Syntax

```
<<identifier>> <<C&A operator>> <<value>>;
```

Examples

```
i += 128; //If i = 32 now it is 160
j %= 2; //If j = 28.0 now it is 0.0
```

20

Basic Output

- System.out.print(<<argument>>);
  - –Prints the argument from cursor's current location.
- System.out.println();
  - –Prints the argument and moves the cursor down one line.
  - –Adds the endline character '\n' to the end.
- Prints to the standard system output, the console.

Syntax

```
System.out.print(<<argument>>);
System.out.println(<<argument>>);
```

Examples

```
int i = 22;
double j = 3.14;
System.out.print(i);
System.out.print("_"+j+" ");
System.out.println(i+" "+j);
System.out.println(i+2);
```

Console

```
22_3.14 22 3.14
24
```

21

Basic Input

- Use Scanner to read from Console
- Must import type Scanner from "java.util" package
  - –import java.util.Scanner;
- Create an instance of type Scanner that "scans" the standard system input
  - –Scanner keyboard = new Scanner(System.in);
- Useful methods
  - –next()
  - –nextLine()
  - –nextInt()
  - –nextDouble()
- Also can be used to "scan" Strings, files, network traffic, etc.

Examples

```
Scanner keyboard = new Scanner(System.in);
String name = keyboard.nextLine();
int i = keyboard.nextInt();
keyboard.nextLine();//Useful "fix-up"
double j = keyboard.nextDouble();
keyboard.nextLine();//Useful "fix-up"
System.out.println(name+ " " + i + " " + j);
```

Console

```
JJ
64
3.14
JJ 64 3.14
```

22

Strings

- Object type
- Array of characters
- Denoted by double quotes ("")
  - –Characters are single quotes (')
- The plus (+) operator concatenates (appends) a value with a String
- Useful methods
  - –charAt(index)
  - –substring(startIndex)
  - –substring(startIndex, endIndex)
  - –toUpperCase()
  - –toLowerCase()
  - –split(regular expression)

Examples

```
String str = "abcdefg";
System.out.println(str.charAt(0));
String str2 = str.substring(2,5);
System.out.println(str2);
```

Console

```
a
cde
```

23

Strings

- Object type
- Array of characters

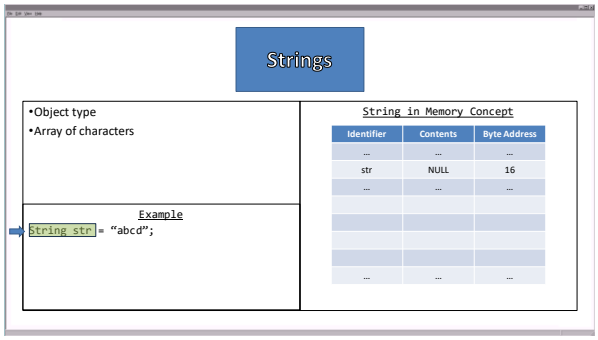
Example

```
String str = "abcd";
```

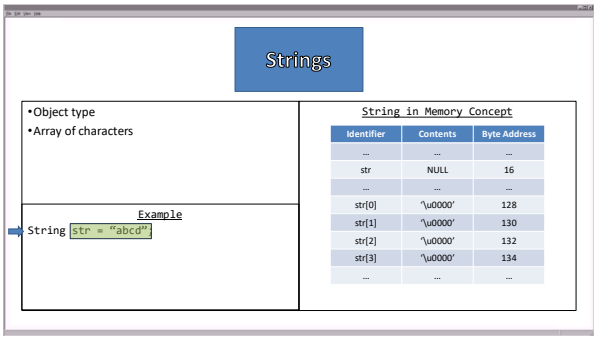
String in Memory Concept

Identifier	Contents	Byte Address
…	…	…
…	…	…

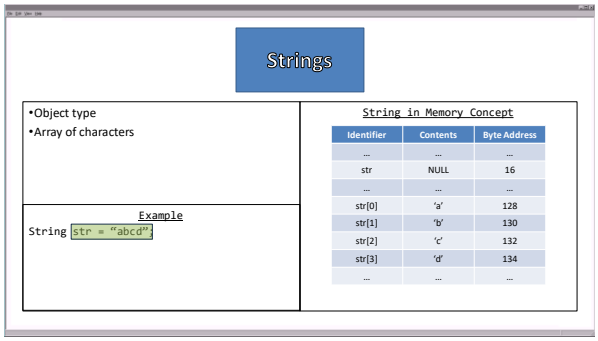
24



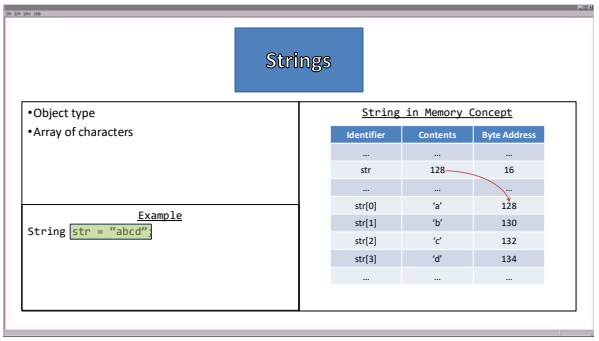
25



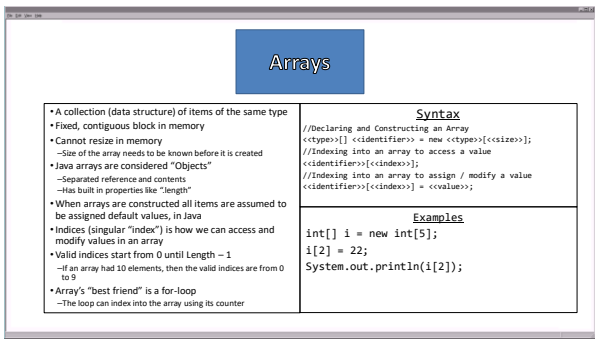
26



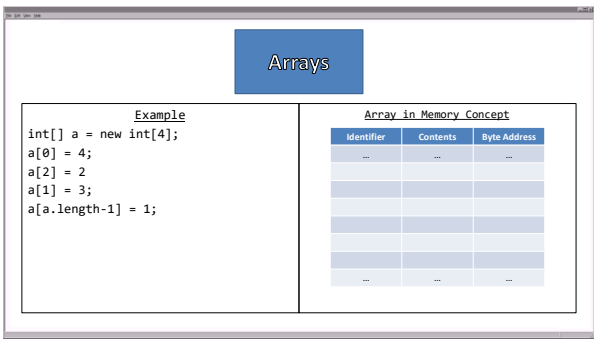
27



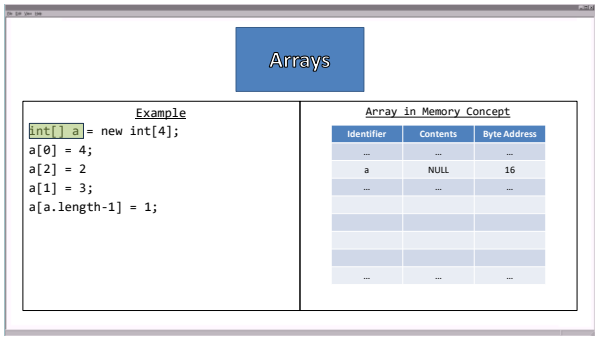
28



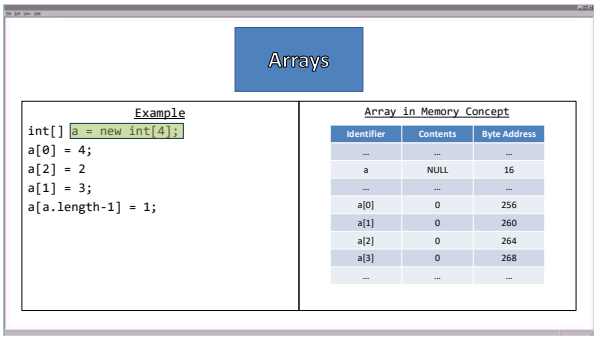
29



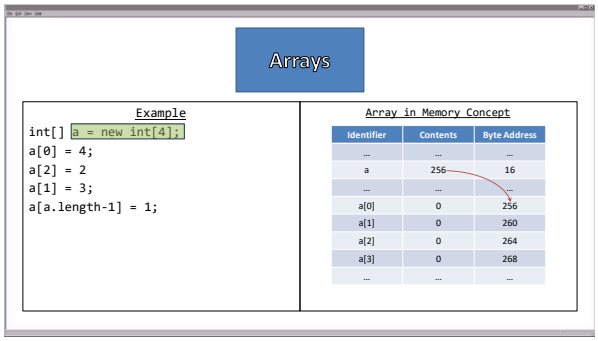
30



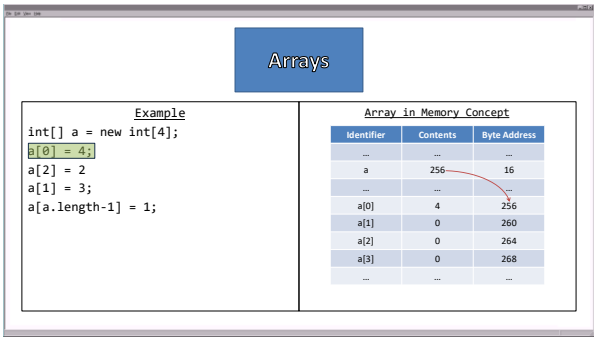
31



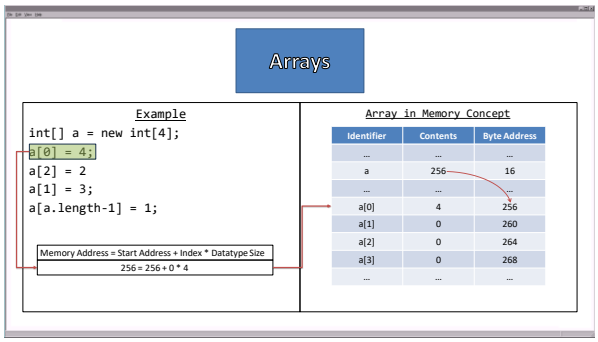
32



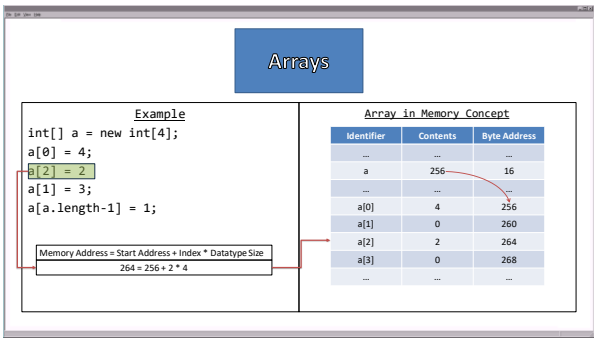
33



34



35



36

Arrays

Example

```
int[] a = new int[4];
a[0] = 4;
a[2] = 2
a[1] = 3;
a[a.length-1] = 1;
```

Memory Address = Start Address + Index * Datatype Size

260 = 256 + 1 * 4

Array in Memory Concept

Identifier	Contents	Byte Address
...	...	...
a	256	16
...	...	...
a[0]	4	256
a[1]	3	260
a[2]	2	264
a[3]	0	268
...	...	...

37

Arrays

Example

```
int[] a = new int[4];
a[0] = 4;
a[2] = 2
a[1] = 3;
a[a.length-1] = 1;
```

Memory Address = Start Address + Index * Datatype Size

260 = 256 + 1 * 4

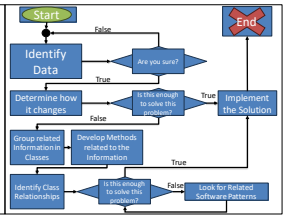
Array in Memory Concept

Identifier	Contents	Byte Address
...	...	...
a	256	16
...	...	...
a[0]	4	256
a[1]	3	260
a[2]	2	264
a[3]	1	268
...	...	...

38

Problem Solving

2. Determine how the data changes over time.
   - This demonstrates how the software *behaves*.
   - What are the noticeable patterns from input to output?
   - Construct Flow-Charts for low-level logic.
   - What are some common *actions* within these patterns? Can they be grouped and reused as functions / methods?
   - What features modify the data and how?
   - Look for the **Verbs**



39

Branching Statements

- If-statement
- If the Boolean expression evaluates "true" then the body of the if-statement is executed, and otherwise is ignored
- Putting curly braces "{}" to denote the body of the if-statement is strongly encouraged
- Do not put a semicolon ";" after the parenthesis
  - It will ignore the Boolean expression
- Spoken
  - "if this is true then do this"

Syntax

```
if(<<Boolean expression>>)
{
      //Body of the if-statement
}
```

Examples

```
if(a == b)
{
   System.out.println("a is equal to b");
}
```

40

Branching Statements

- Else-statement
- Requires a proceeding if-statement
  - If-statements do not require an else-statement
- If the Boolean expression is "false" then the body of the else-statement is executed, and otherwise is ignored
- Putting curly braces "{}" to denote the body of the else-statement is strongly encouraged
- Spoken:
  - "if this is true then do this, otherwise (else) do that"

Syntax

```
if(<<Boolean expression>>)
{      //Body of the if-statement
}
else
{      //Body of the else-statement
}
```

Examples

```
if(a == b)
{
   System.out.println("a is equal to b");
}
else
{
   System.out.println("a is not equal to b");
}
```

41

Branching Statements

- Else-If-statement
- Shorthand for an if-statement inside the body of an else-statement that connected to preceding If-statement
  - "If nested within the else of another if"
- Requires a proceeding if-statement or else-if-statement
- Putting curly braces "{}" to denote the body of the else-statement is strongly encouraged

Syntax

```
if(<<Boolean expression>>)
{      //Body of the if-statement
}
else if(<<Boolean expression>>)
{      //Body of the else-if-statement
}
```

Examples

```
if(a < b)
{
   System.out.println("a is less than b");
}
else if(a > b)
{
   System.out.println("a is greater than b");
}
else
{
   System.out.println("a and b are equal);
}
```

42

Branching Statements

```
if (Boolean_Expression_1)
{
   Statement_1
}
else if (Boolean_Expression_2)
{
   Statement_2
}
else if (Boolean_Expression_3)
{
   Statement_3
}
else
{
   Default_Statement
}
```

```
if (Boolean_Expression_1)
{
   Statement_1
}
else
{
   if (Boolean_Expression_2)
   {
      Statement_2
   }
   else
   {
      if(Boolean_Exp_3)
      {
         Statement_3
      }
      else
      {
         Default_Statement
      }
   }
}
```

43

Branching Statements

```
if (Boolean_Expression_1)
{
   Statement_1
}
else if (Boolean_Expression_2)
{
   Statement_2
}
else if (Boolean_Expression_3)
{
   Statement_3
}
else
{
   Default_Statement
}
```

```
if (Boolean_Expression_1)
{
   Statement_1
}
if (Boolean_Expression_2)
{
   Statement_2
}
if (Boolean_Expression_3)
{
   Statement_3
}
else
{
   Default_Statement
}
```

44

Boolean Expressions

- True or False Value
- Common Boolean Operators
  - –"==" : Equal to
  - –"!=" : Not Equal
  - –"<" : strictly less than
  - –">" : strictly greater than
  - –"<=": less than or equal to
  - –">=": greater than or equal to

Syntax

```
<<value>> <<Boolean operator>> <<value>>;
```

Examples

```
boolean a = 12 > 3;
if(a)//Or a == true
{
   System.out.println("Here");
}
else
{
   System.out.println("Not here");
}
```

45

Compound Boolean Expressions

- Combines multiple Boolean expressions
- Common Compound Boolean Expression Operators
  - –"&&" : AND – both must be true to evaluate true
  - –"||" : OR – one or both be true to evaluate true
  - –"!" : NOT – negates the value. Not a binary operator like AND or OR
- Truth Table

A	B	A && B	A || B
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE

Syntax

```
<<Boolean expression>> <<operator>> <<Boolean expression>>;
```

Examples

```
boolean a = 2 != 0 && 12 > 3;
if(a)//Or a == true
{
   System.out.println("Here");
}
else
{
   System.out.println("Not here");
}
```

46

Loops

- Runs a block of code some number of times
- The body of the loop runs while a Boolean expression is true
- While-loops are "check then iterate"
  - –The body of the loop may run 0 to many times
- Great when unsure how many times the loop needs to run
- Spoken:
  - –"While this is true, keep doing this"
  - –"Until this is false, keep doing this"

Syntax

```
while(<<Boolean Expression>>)
{
      <<Body of the Loop>>
}
```

Examples

```
while(!gameOver)
{
   gameLoop();
}
```

47

Loops

- Do-While-loops are "iterate then check"
  - –The body of the loop may run 1 to many times
- Great when unsure how many times the loop needs to run, but also needs to run the body of the loop at least once.
- Semi-colon must come after the while or it's a syntax error
- Spoken:
  - –"Do this While this is true"

Syntax

```
do
{
      <<Body of the Loop>>
}while(<<Boolean Expression>>);
```

Examples

```
do
{
   eat();
}while(full == false);
```

48

Loops

- For-loops are "counting loops"
  - The body of the loop runs some number of times.
- Great when it is known how many times the loop must run
- The sequence of a for-loop goes as
  1. Initialize counter
  2. Check Boolean Expression
  3. Run Body of the Loop
  4. Update the counter and go back to Step 2.
- Spoken:
  - "For this starts here and ends here, do this that many times"

Syntax

```
for(<<initialize counter>>;<<Boolean expression>>;<<update counter>>)
{
    <<Body of the Loop>>
}
```

Examples

```
for(int i=0;i<10;i++)
{
  System.out.println(i);
}
```

49

Arrays

Example

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
  a[i] = i*2;
}
```

Array in Memory Concept

Identifier	Contents	Byte Address
...	...	...

50

Arrays

Example

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
  a[i] = i*2;
}
```

Header

Body

Array in Memory Concept

Identifier	Contents	Byte Address
...	...	...
a	512	16
...	...	
Meta Data	Class / Datatype	512
a.length	4	520
a[0]	0	524
a[1]	0	528
a[2]	0	532
a[3]	0	536
...	...	...

51

Arrays

Example

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
  a[i] = i*2;
}
```

Array in Memory Concept

Identifier	Contents	Byte Address
...	...	...
a	512	16
i	0	20
...	...	
Meta Data	Class / Datatype	512
a.length	4	520
a[0]	0	524
a[1]	0	528
a[2]	0	532
a[3]	0	536
...	...	...

52

Arrays

Example

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
  a[i] = i*2;
}
```

Array in Memory Concept

Identifier	Contents	Byte Address
...	...	...
a	512	16
i	**0**	20
...	...	
Meta Data	Class / Datatype	512
a.length	**4**	520
a[0]	0	524
a[1]	0	528
a[2]	0	532
a[3]	0	536
...	...	...

53

Arrays

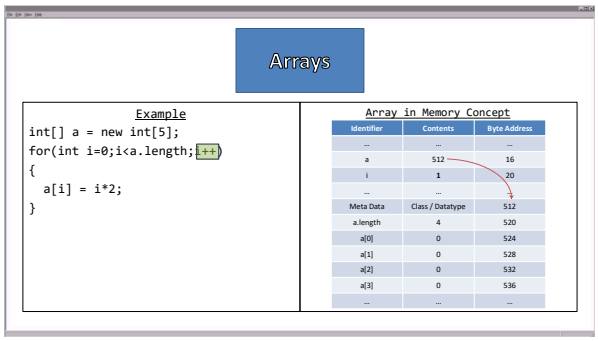
Example

```
int[] a = new int[5];
for(int i=0;i<a.length;i++)
{
  a[i] = i*2;
}
```

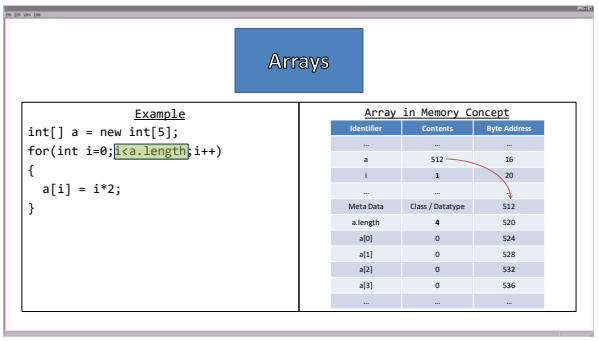
Array in Memory Concept

Identifier	Contents	Byte Address
...	...	...
a	512	16
i	0	20
...	...	
Meta Data	Class / Datatype	512
a.length	4	520
a[0]	**0**	524
a[1]	0	528
a[2]	0	532
a[3]	0	536
...	...	...

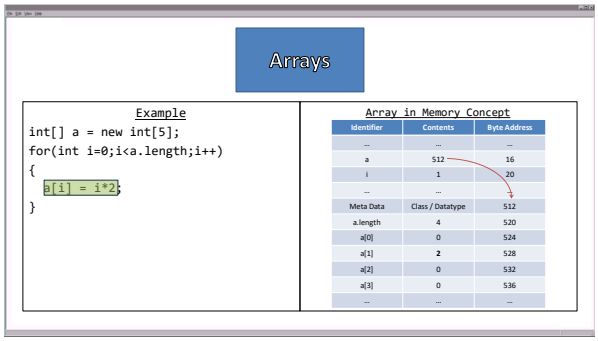
54



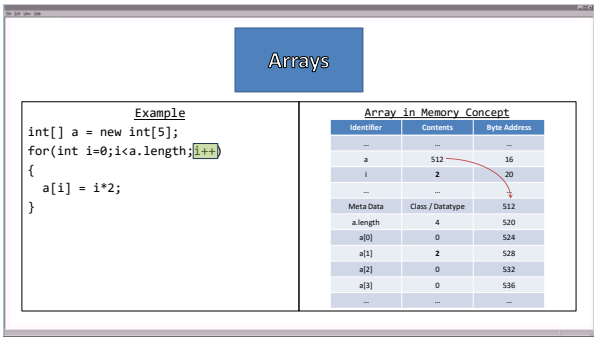
55



56

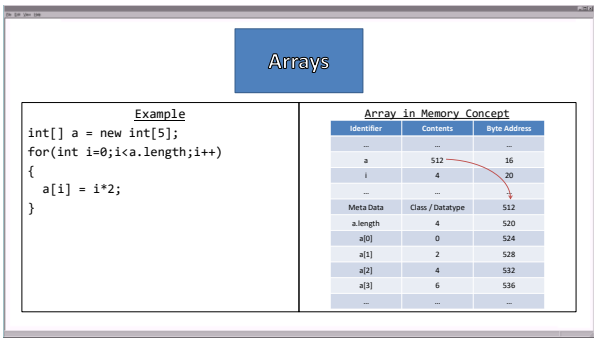


57



58

59



60

Arrays

Example

```
int[] a = new int[5];
for(int i=a.length-1;i>=0;i--)
{
  a[i] = i*2;
}
```

Array in Memory Concept

Identifier	Contents	Byte Address
...	...	...
a	512	16
i	0	20
...	...	
Meta Data	Class / Datatype	512
a.length	4	520
a[0]	0	524
a[1]	2	528
a[2]	4	532
a[3]	6	536
...	...	...

61

Programming Review Part 02

62