FORMATTING

\t - Insert a tab in the text at this point.
\b - Insert a backspace in the text at this point.
\n - Insert a newline in the text at this point.
\r - Insert a carriage return in the text at this point.
\' - Insert a single quote character in the text at this point.
\" - Insert a double quote character in the text at this point.
\\ - Insert a backslash character in the text at this point.

CONVERSIONS

int → String	| String.valueOf(int)
String → int	| Integer.parseInt(String)
String → Integer	| Integer.valueOf(String)
double → String	| String.valueOf(double)
String → double	| Double.parseDouble(String)
char → String	| String.valueOf(char)
String → char	| String.charAt(index)
Double → Int	| (int) data
Char → ASCII	| char character = 'A';

int asciiValue = (int) character;

WIDENING PRIMITIVE CONVERSION

byte	→	short, int, long, float, or double
short	→	int, long, float, or double
char	→	int, long, float, or double
int	→	long, float, or double
long	→	float or double
float	→	double

STRINGBUFFER METHODS

```
.insert(int index, String str)
.replace(int start, int end, String str)
.delete(int start, int end)
.deleteCharAt(int index)
.reverse()
.charAt(int index)
.setCharAt(int index, char ch)
```

ARRAY METHODS

```
Arrays.copyOf(original, newLength);
Arrays.copyOfRange(original, from, to);
Arrays.sort(array);
Arrays.asList(myList))      Converting an Array to an ArrayList
```

SWITCH CASE

```
switch(grade) {
    case "A" :
            System.out.println("Excellent!");
            break;
    case "B" :
            System.out.println("Well done!");
            break;
    case "C" :
            System.out.println("You passed!");
            break;
    default :
            System.out.println("Invalid grade!");
            break;
    }
```

CONDITIONAL OPERATOR ?:

```
variable = (condition) ? expression1 : expression2
```

SPLITTING

```
import java.util.regex.*;
www.google.com → myString.split("\\.");
```

RETRIEVE ENUM CONSTANT USING PARAMETER

```
public static ShapeType getShapeType(int numberOfSides) {
    for (ShapeType shape : ShapeType.values()) {
        if (shape.getNumberOfSides() == numberOfSides) {
            return shape;
        }
    }
    return null;
```

INVOKING METHODS

Valid: If method is called on a SUPER type and method exists in SUPER reference OR if method is called on CHILD type, and method exists in child/superclass
Output: Uses the ACTUAL type.
RUN-TIME vs COMPILE TIME ERRORS:
COMPILE TIME: Type errors
RUN-TIME: Incorrect conversion

OTHER METHODS

```
for (String value : words) {char letter = Character(value);      Iterating through Array of strings
(char c : word.toCharArray()) {      if (Character.isDigit(c)){      Iterating through a String
```

ENUM

```
PizzaSize.SMALL.getPrice());      Calling a method on an ENUM
PizzaSize s = PizzaSize.LARGE      Creating an ENUM object
```

KEYWORDS

public → The class, method, or field is accessible from any other class in any package
protected → The method or field is accessible within the same package and by subclasses (even if they are in different packages).
private → The method or field is accessible only within the same class where it is declared.
static → the method or field belongs to the class itself rather than instances of the class. It can be accessed without creating an object of the class.
final → the value of a variable cannot be changed, a method cannot be overridden, or a class cannot be subclassed.
abstract → a class or method is incomplete and must be implemented by subclasses. An abstract class cannot be instantiated.

DO-WHILE LOOP

```
do {
   System.out.println("Enter a multiple of 3:");
    number = scanner.nextInt();
} while (number % 3 != 0);
```

```
class ClassOne{
        public void methodOne(){
        System.out.println("Method One, Class One");}
        public int methodTwo(){ return 5; }
        public void methodThree(){System.out.println("Method Three, Class One");}}

class ClassTwo extends ClassOne{
        public void methodThree(){System.out.println("Method Three, Class Two");}}

class ClassThree extends ClassTwo{
        public int methodTwo(){ return 10;}}

class ClassFour extends ClassOne{
        public int methodTwo() { return super.methodTwo() + 5;}
        public void methodFour(){System.out.println("Method Four, Class Four");}
        public void methodFour(int x) {System.out.println("Method Four(x), Class Four");}}
```

OVERLOADED METHODS:
methodFour(), methodFour(int x)
OVERRIDDEN METHODS:
methoTwo, methodThree
LEGAL ASSIGNMENT EG:

```
ClassOne c3 = new ClassThree();
```

ILLEGAL ASSIGNMENT EG:

```
ClassThree c2 = new ClassFour();
ClassTwo c1 = new ClassOne();
```

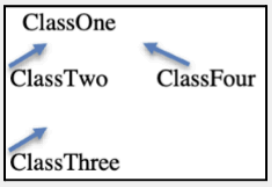
COMPILE TIME ERROR:

```
ClassOne c4 = new ClassFour();
c4.methodFour();
```

(No methodFour in ClassOne)
NO COMPILE TIME ERROR:

```
ClassOne c2 = new ClassTwo();
c2.methodThree();
ClassOne c3 = new ClassThree();
c3.methodThree();
```

CLASS HIERARCHY:



RUN TIME ERROR:

```
ClassOne c4 = new ClassFour();
((ClassThree) c4).methodOne();
```

(Can't convert ClassFour into ClassThree)
NO RUN TIME ERROR:

```
ClassOne c4 = new ClassFour();
((ClassFour) c4).methodFour();
ClassOne c3 = new ClassThree();
((ClassThree) c3).methodOne();
```

(ITERATOR) CLASS EXTENDS PARENT

```
class Garden extends GardenBlock implements Iterable<GardenBlock> {
   private ArrayList<GardenBlock> blocks = new ArrayList<GardenBlock>();
   public Garden() { super(100, 100); }
   public void addGardenBlock(GardenBlock b){ …}
   public int getGardenArea(){ …}
   public String toString() { …}
   public Iterator<GardenBlock> iterator() {
      Collections.sort(blocks);   // Sorts list before iteration
      return new BlockIterator(blocks); }
}
class BlockIterator implements Iterator<GardenBlock> {
   private int nextIndex = 0;
   private ArrayList<GardenBlock> blocks;
   public BlockIterator(ArrayList<GardenBlock> myList) {
      this.blocks = myList; }
   public boolean hasNext() {return nextIndex < blocks.size();}
   public GardenBlock next() {
      if (!hasNext()) {throw new NoSuchElementException();}
      return blocks.get(nextIndex++); }}
```

(ITERATOR) NO EXTENSION

```
class BasicUniqueEven implements Iterable<Integer>{
   private ArrayList<Integer> items = new ArrayList<Integer>();
   public BasicUniqueEven() {}
   public boolean add(int number) { …}
   public void addAll(int[] array){ …}
   public void removeByValue(int number){ …}
   public boolean contains(int number) { …}
   public String toString() { …}
      // Extra methods to support iteration
   public int size() {return items.size();}
   public Integer get(int index) {return items.get(index);}
   public Iterator<Integer> iterator() {return new UniqueEvenIterator(this); }
}
class UniqueEvenIterator implements Iterator<Integer> {
   private int nextIndex = 0;
   private BasicUniqueEven list;
   public UniqueEvenIterator(BasicUniqueEven list) {this.list = list; }
   public boolean hasNext() {return nextIndex < list.size(); }
   public Integer next() {return list.get(nextIndex++); }}
```

ITERATE THROUGH 2D ARRAY

```
for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array[i].length; j++) {
           System.out.print(array[i][j] + " ");
        }
```

MODIFY ARRAYLIST IN PLACE

```
for (int i = 0; i < list.size(); i++) {
        list.set(i, list.get(i) * 2);
    }
```

SIMPLE ENUM

```
enum ShapePerimeter {
          TRIANGLE(3),
          SQUARE(4),
          PENTAGON(5),
          private int numberOfSides;
          private ShapePerimeter(int sides) {this.numberOfSides= sides;}
          public int getPerimeter(int sideLength) {return numberOfSides *
sideLength;}
```

ShapePerimeter.ordinal() → returns position of enum object
object.name() → returns name of enum
(TRIANGLE/SQUARE/PENTAGON)

getIndexOfLargestOdd Method

```
public static int getIndexOfLargestOdd(int[] numbers) {
  int largestOdd = -1;
  int index = -1;
  for (int value: numbers){if (value%2 == 1 && value > largestOdd) {
largestOdd = value; } }
  for (int i=0; i<numbers.length; i++){ if (numbers[i] == largestOdd) {return
i;}}
  return -1;}
```

```
interface FinancialAidEligible { }
abstract class Person {
  int ID;
  public int getID() { return ID; }
}
class Student extends Person {}
class Undergraduate extends Student implements FinancialAidEligible {}
```

FinancialAidEligible p1 = new Undergraduate(); **Legal**
The Undergraduate class implements the FinancialAidEligible interface. This means that an Undergraduate object can be assigned to a variable of type
FinancialAidEligible p2 = new FinancialAidEligible(); **Compile-time error**
FinancialAidEligible is an interface, and you cannot instantiate an interface directly in Java.
FinancialAidEligible p3 = new Student(); **Compile-time error**
The Student class does not implement the FinancialAidEligible interface
FinancialAidEligible[] people = new FinancialAidEligible[10]; **Legal**
In Java, you can create an array of any reference type, including interfaces.

OTHER ARRAYLIST METHODS

subList(int fromIndex, int toIndex): Returns a view of the portion of the list between fromIndex, inclusive, and toIndex, exclusive.
remove(int index): Removes the element at the specified position in the list.
remove(Object o): Removes the first occurrence of the specified element from the list.
set(int index, E element): Replaces the element at the specified position with the specified element.
toArray(): Returns an array containing all elements in the list.
isEmpty(): Returns true if the list contains no elements.
get(int index): Returns the element at the specified position in the list.
indexOf(Object o): Returns the index of the first occurrence of the specified element, or -1 if not found.
lastIndexOf(Object o): Returns the index of the last occurrence of the specified element, or -1 if not found.
add(int index, E element): Inserts the specified element at the specified position in the list.