JAVA BOOTCAMP

TEXT AND BINARY FILES

TEXT AND BINARY FILES

TOPICS - WORKING WITH FILE I/O

TEXT AND BINARY FILES

A PACKAGE FOR WORKING WITH DIRECTORIES AND FILES

```
java.nio.*
java.io.*
```

IMPORTANT FILE I/O TYPES FILE IO CLASSES AND INTERFACES PATH PATHS FILE FILES

TEXT AND BINARY FILES

METHODS OF THE *PATH* INTERFACE

- getFileName()
- getParent()
- getRoot()
- toAbsolutePath()

TEXT AND BINARY FILES

METHODS OF THE *FILES* CLASS

- exists(Path)
- isReadable(Path)
- isWritable(Path)
- isDirectory(Path)
- size(Path)
- createFile(Path)
- createDirectory(Path)
- delete(Path)

TEXT AND BINARY FILES

CODE THAT CREATES A DIRECTORY IF IT DOESN'T ALREADY EXIST

```
String dirString = "c:/murach/java/files";
Path dirPath = Paths.get(dirString);
if (Files.notExists(dirPath)) {
    Files.createDirectories(dirPath);
}
```

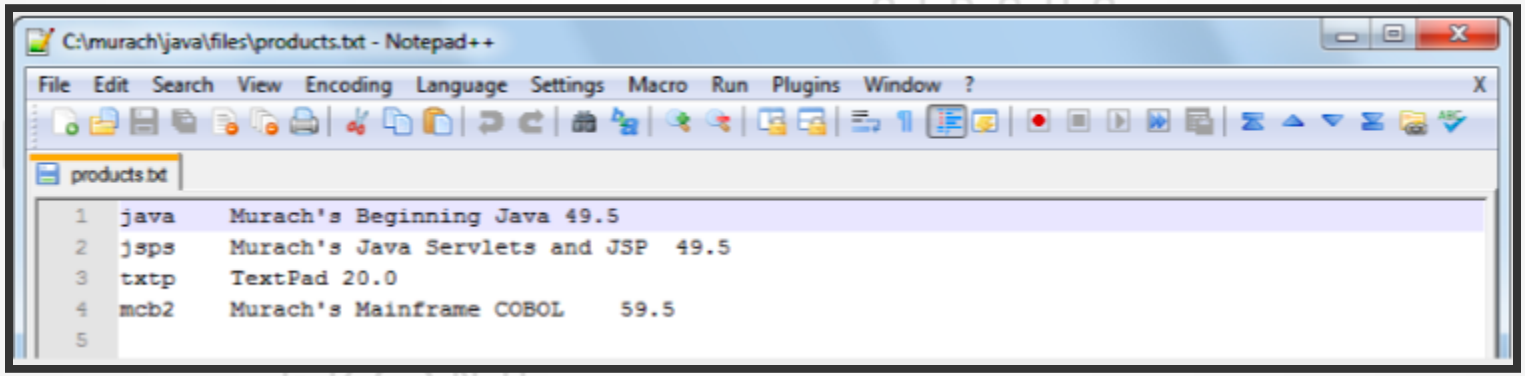
TEXT AND BINARY FILES

CODE THAT CREATES A FILE IF IT DOESN'T ALREADY EXIST

```
String fileString = "products.txt";
Path filePath = Paths.get(dirString, fileString);
if (Files.notExists(filePath)) {
    Files.createFile(filePath);
}
```

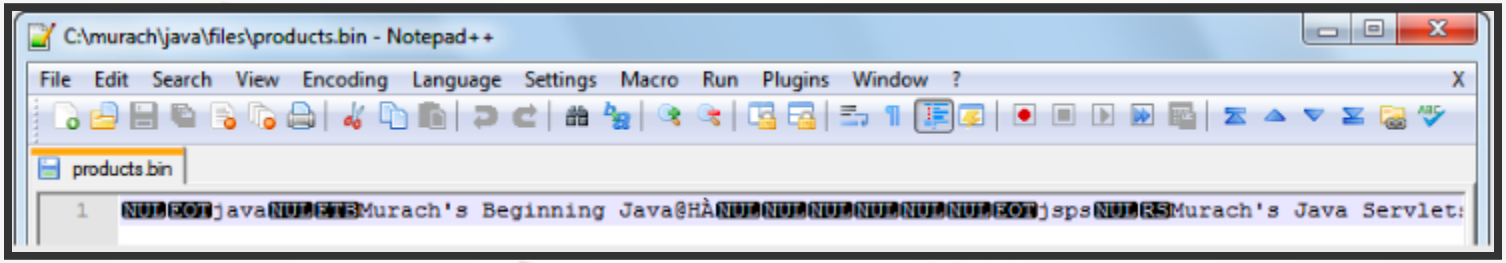
TEXT AND BINARY FILES

A TEXT FILE THAT'S OPENED BY A TEXT EDITOR



TEXT AND BINARY FILES

A BINARY FILE THAT'S OPENED BY A TEXT EDITOR



TEXT AND BINARY FILES

TWO TYPES OF FILES - TEXT - BINARY

TWO TYPES OF STREAMS - CHARACTER - BINARY

TEXT AND BINARY FILES

WRITE DATA TO THE FILE

```
Path productsPath = Paths.get("products.txt");
File productsFile = productsPath.toFile();

try {
    PrintWriter out = new PrintWriter(productsFile);
    out.println("java\tMurach's Beginning Java\t49.50");
}
catch (IOException e)
{
    System.out.println(e);
}
```

TEXT AND BINARY FILES

READ DATA FROM THE FILE

```
Path productsPath = Paths.get("products.txt");
File productsFile = productsPath.toFile();
try {
    FileReader r = new FileReader(productsFile);
    BufferedReader in = new BufferedReader(r);
    ...
```

TEXT AND BINARY FILES

READ DATA FROM THE FILE(CONTINUE)

```
        String line = in.readLine();
        while (line != null)
        {
            System.out.println(line);
            line = in.readLine();
        }
    } catch (IOException e) {
        System.out.println(e);
    }
```

TEXT AND BINARY FILES

- IOException
- EOFException
- FileNotFoundException

RECAP

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT

- Know File I/O Java classes
- Know the functions of the different File I/O classes
- Using File I/O classes to get information about files and directories
- Know file types and the differences between them
- How to write and read from files

THREADS

THREADS

TOPICS

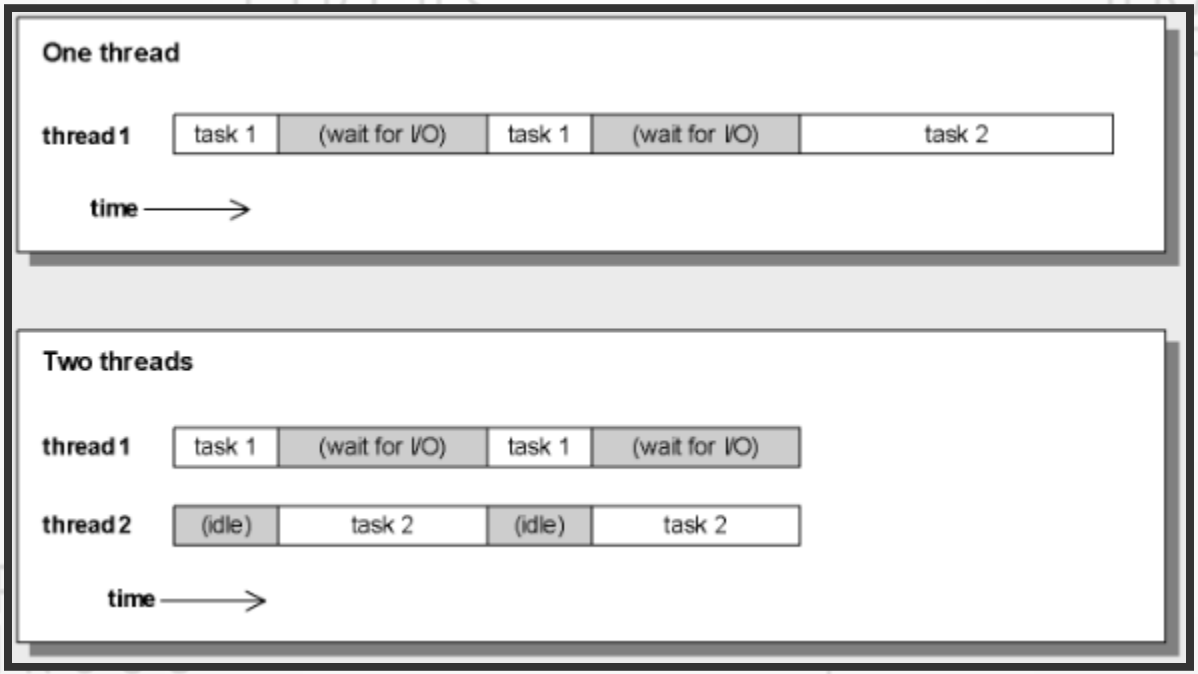
- What are threads
- Threads' lifecycle

THREADS

Threads are parts of the program set to run independent of the rest of the program. We can segregate processor-intensive functions in a Java class to run separately from the rest of the programs using *threads*. This is also referred to as *multitasking*.

THREADS

HOW USING THREADS CAN IMPROVE PERFORMANCE



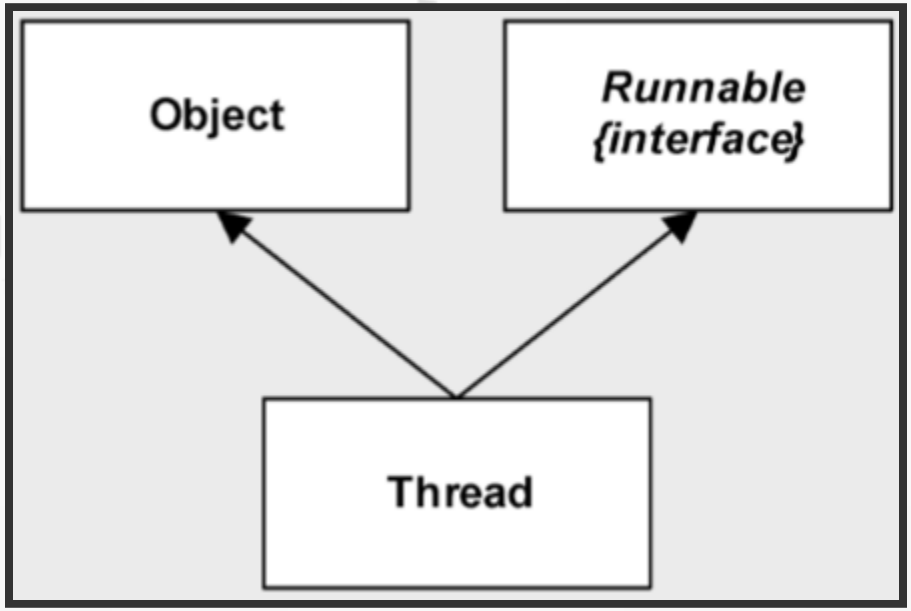
THREADS

TYPICAL USES FOR THREADS

- To improve the performance of applications with extensive I/O operations
- To improve the responsiveness of GUI applications
- To allow two or more users to run server-based applications simultaneously

THREADS

CLASSES AND INTERFACES USED TO CREATE THREADS



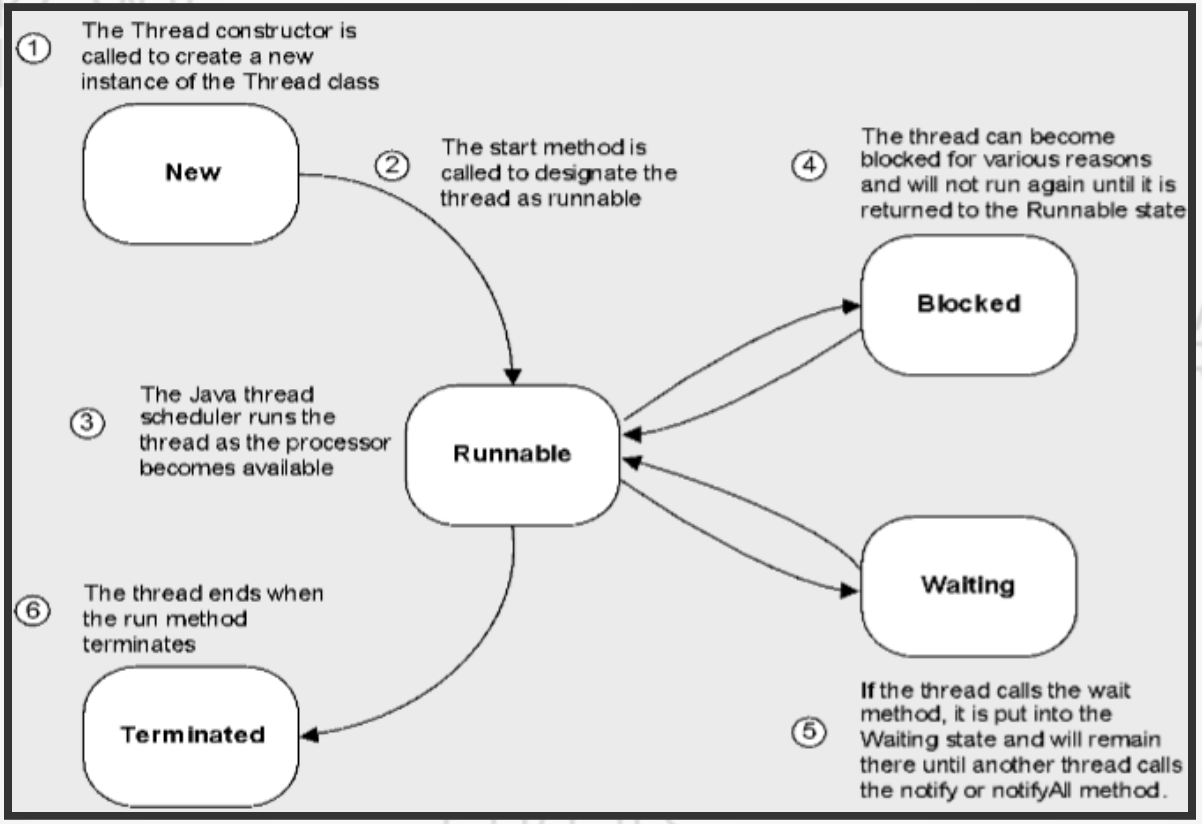
THREADS

TWO WAYS TO CREATE A THREAD

- Inherit the Thread class.
- Implement the Runnable interface, then pass a reference to the Runnable object to the constructor of the Thread class. This is useful if the thread needs to inherit a class other than the Thread class.

THREADS

LIFE CYCLE OF THREADS



THREADS

THE THREAD CLASS

```
java.lang.Thread;
```

THREADS

COMMON METHODS OF THE THREADS CLASS

- start()
- getName()
- currentThread()
- sleep(long)
- interrupt()

THREADS

CREATING A THREAD FROM THE THREADS CLASS

1. Create a class that inherits the Thread class.
2. Override the run method to perform the desired task.
3. Create the thread by instantiating an object from the class.
4. Call the start method of the thread object.

THREADS

CREATING A THREAD FROM A RUNNABLE INTERFACE

1. Create a class that implements the Runnable interface.
2. Implement the run method to perform the desired task.
3. Create the thread by supplying an instance of the Runnable class to the Thread constructor.
4. Call the start method of the thread object.

THREADS

A MAIN CLASS THAT STARTS A THREAD

```
public class Main
{
    public static void main(String[] args)
    {
        Thread t1 = Thread.currentThread();
        System.out.println(t1.getName() + " started.");
         Thread t2 = new Thread(new Task());  // create the new thread
         t2.start(); // start the new thread
        System.out.println(t1.getName() + " starts " +
                t2.getName() + ".");
        System.out.println(t1.getName() + " finished.");
    }
}
```

RECAP

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT:

- What are threads?
- Where are threads used?
- How threads can enhance performance
- Classes and interfaces used in thread programming
- Thread lifecycle
- Ways to create and use threads

ADVANCED JAVA

DESIGN PATTERNS

LAMBDA EXPRESSIONS

LAMBDA EXPRESSIONS

TOPICS

- What are Lambda Expressions?
- How to use Lambda Expressions

LAMBDA EXPRESSIONS

LAMBDA EXPRESSSIONS

WHAT ARE LAMBDA EXPRESSIONS?

- *Lambda expressions* are based on Lambda Calculus.
- A *Lambda expression* can be looked at as a program which when evaluated returns a result consisting of another lambda expression.
- A Java lambda is basically a method without a declaration usually written as (parameters) -> { body }.

LAMBDA EXPRESSIONS

GENERAL RULES

- A lambda can have zero or more parameters separated by commas and their type can be explicitly declared or inferred from the context.
- Parenthesis are not needed around a single parameter.
- ( ) is used to denote zero parameters.
- The body can contain zero or more statements.
- Braces are not needed around a single-statement body.

HOW TO USE LAMBDA EXPRESSIONS

EXAMPLE

EXAMPLE

```
List<Integer> intSeq = Arrays.asList(1,2,3);
intSeq.forEach(x -> System.out.println(x));
```

x -> System.out.println(x) is a lambda expression that defines an anonymous function with one parameter named x

EXAMPLE

EXAMPLE

```
List< Integer> intSeq = Arrays.asList(1,2,3);
intSeq.forEach(x -> {
   int y = x * 2;
   System.out.println(y);
});
```

You can define local variables inside the body of a lambda expression.

EXAMPLE

EXAMPLE

```
List< Integer> intSeq = Arrays.asList(1,2,3);

intSeq.forEach((Integer x -> {
   x += 2;
   System.out.println(x);
});
```

You can, if you wish, specify the parameter type.

LAMBDA EXPRESSIONS

HOW DOES JAVA DEAL WITH LAMBDA EXPRESSIONS?

- JAVA converts a lambda expression into a function that can be called later.
- For example, *x -> System.out.println(x)* could be converted into a generated static function

```
public static void printX(Integer x) {
    System.out.println(x);
}
```

EXAMPLES

More examples!

RECAP

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT:

- What are Lambda Expressions.
- How to use Lambda Expressions.