Lecture 37

- Covers
  - Traditional Java input

- Reading: Savitch Appendix 4

Standard Java console input

- System.out is an object (an instance of the PrintStream class) that has methods to write to the console window
- System.out is designed to output values with a range of types (String, int, double, etc.)
- Console input is read from the System.in object (an instance of the InputStream class)
- System.in can only read bytes

InputStreamReader class

- To read input as characters, we need a different type of object: an instance of the InputStreamReader class

```
InputStreamReader charReader = new
                               InputStreamReader(System.in);
```

InputStreamReader class

- Methods
  - InputStreamReader( )
    - Constructor takes an InputStream object
  - read( )
    - Reads a character
    - Returns an integer
      - which is the value of the character read
      - or -1 if the end of the input stream has been reached
- Package
  - Part of the java.io package

InputStreamReader example

```
import java.io.*;
public class ISRTest
{
    public static void main(String[ ] args) throws IOException
    {
        InputStreamReader isr = new InputStreamReader(System.in);

        System.out.println("Enter a line of text:");
        String input = "";
        char temp = (char) isr.read( );
        while (temp != '\n')
        {
            input += temp;
            temp = (char) isr.read( );
        }
        System.out.println("You entered: \"" + input + "\"");
    }
}
```

Error handling in the read( ) method

- When there is a problem in the read( ) method of the InputStreamReader class, the problem is dealt with through Java's exception handling mechanism

Exception handling

- When something unusual happens in a method, Java (or the method code itself) may signal the problem using an exception
- An exception is an object that is created and the process of signalling the error is called "throwing an exception"
- When an exception is thrown, the problem can be dealt with in a separate piece of code that "handles" the exception

Exception handling

- The read( ) method of the InputStreamReader class may throw an exception
- The type of exception object it may throw is an IOException
- The read( ) method does not have code to handle the exception, but it is propagated back to where the read( ) method is called

Exception handling

- How do we deal with the exception propagated back to the calling method?
  - Handle the exception
  - Indicate that the exception is not handled in the calling method and that the exception will be propagated back from this method

```
public static void main(String[ ] args) throws IOException
```

BufferedReader class

- An InputStreamReader reads one character at a time
- To read a string of characters at once we need a different type of object again: an instance of the BufferedReader class
- We can create a BufferedReader object from an InputStreamReader object

```
BufferedReader stringReader = new
                        BufferedReader(charReader);
```

BufferedReader class

- Methods
  - BufferedReader( )
    - Constructor takes an InputStreamReader object (or other types of Readers)
  - read( )
    - Reads a character
    - Returns an integer
      - which is the value of the character read
      - or -1 if the end of the input stream has been reached
    - May throw an IOException

BufferedReader class

- Methods
  - readLine( )
    - Reads a line of text up to and including the end of line character
    - Returns a String
      - containing the line's contents (but not the line terminator)
      - or null if the end of the input stream has been reached
    - May throw an IOException
- Package
  - Part of the java.io package

BufferedReader example

```
import java.io.*;

public class BRTest
{
    public static void main(String[ ] args) throws IOException
    {
        InputStreamReader charReader = new InputStreamReader(System.in);
        BufferedReader stringReader  = new BufferedReader(charReader);

        String line1 = "",  line2 = "";

        System.out.println("Enter two lines of text: ");
        line1 = stringReader.readLine( );
        line2 = stringReader.readLine( );

        System.out.println("Line 1: \"" + line1 + "\"");
        System.out.println("Line 2: \"" + line2 + "\"");
    }
}
```

BufferedReader

- We can combine the two declarations needed to create a BufferedReader

```
InputStreamReader charReader = new InputStreamReader(System.in);
BufferedReader stringReader  = new BufferedReader(charReader);
```

into a single declaration

```
BufferedReader stringReader  = new BufferedReader (new
                                    InputStreamReader(System.in));
```

Reading an integer

- Sometimes the text we read as a line may be text representing a number, e.g. “123”
- To use this string as an integer, it has to be converted to its integer form
- The Integer class provides a method to do this
- int Integer.parseInt(String s)
  - Takes a String object, returns the int equivalent
  - Throws a NumberFormat exception if the string does not represent a valid integer

Integer example

```
import java.io.*;

public class ReadIntegerTest
{
    public static void main(String[ ] args) throws IOException
    {
        InputStreamReader charReader = new InputStreamReader(System.in);
        BufferedReader stringReader  = new BufferedReader(charReader);

        String line1 = "",  line2 = "";

        System.out.println("Enter two integers on separate lines: ");
        line1 = stringReader.readLine( );
        line2 = stringReader.readLine( );

        int number1 = Integer.parseInt(line1);
        int number2 = Integer.parseInt(line2);
        System.out.println(line1 + " + " + line2 + " = " + (number1 + number2));
    }
}
```

Reading a double

- As for integers, there is a wrapper class for double values that provides a method called ParseDouble( )
- ParseDouble takes a String argument and returns the double value that the string represents

Double example

```
import java.io.*;

public class DoubleTest
{
    public static void main(String[ ] args) throws IOException
    {
        InputStreamReader charReader = new InputStreamReader(System.in);
        BufferedReader stringReader  = new BufferedReader(charReader);

        String line1 = "",  line2 = "";

        System.out.println("Enter two doubles on separate lines: ");
        line1 = stringReader.readLine( );
        line2 = stringReader.readLine( );

        double number1 = Double.parseDouble(line1);
        double number2 = Double.parseDouble(line2);
        System.out.println(line1 + " + " + line2 + " = " + (number1 + number2));
    }
}
```

Breaking up lines of text

- Sometimes a line of text contains more than one value

  “32 45.1 19”

- Before we can call the parseInt or parseDouble methods we have to separate the String into separate parts
- We could use the substring method to obtain the different parts of the string

Breaking up lines of text

- Given

```
String s = "32 45.1 19"
String t = s.substring(0, 2)
```

- t would be contain the string “32”
- But how do we know how many characters should form each part?
- We could search for the space characters or use the StringTokenizer class

StringTokenizer class

- In package java.util
- The StringTokenizer class allows us to separate a string into tokens
- Tokens are sequences of characters separated by delimiter characters
- The default delimiters are space, tab, newline, and carriage return

StringTokenizer class

- Selected methods

```
public StringTokenizer(String s)
public StringTokenizer(String s, String delims)
// The above constructor allows us to specify which characters
// we want to be delimiters
public int countTokens( )
public boolean hasMoreTokens( )
public String nextToken( )
public String nextToken(String delim)
```

Example 1

- Let String s be a sentence. The following code segment extracts the words one by one and displays them on a separate line

```
StringTokenizer tokenizer = new StringTokenizer(s);
while (tokenizer.hasMoreTokens( ))
{
    String word = tokenizer.nextToken( );
    System.out.println(word);
}
```

Example 2

- Let s be a string containing product number, description and price as shown below

  "P10: Computer desk $95.99"

  Write a code segment to extract the three components and display each on a separate line

```
StringTokenizer tokenizer = new StringTokenizer(s, ":$");
String prodNr = tokenizer.nextToken( );
String description = tokenizer.nextToken( );
String price = tokenizer.nextToken( );  // can convert into double
// output statements
```

Next lecture

- Revision