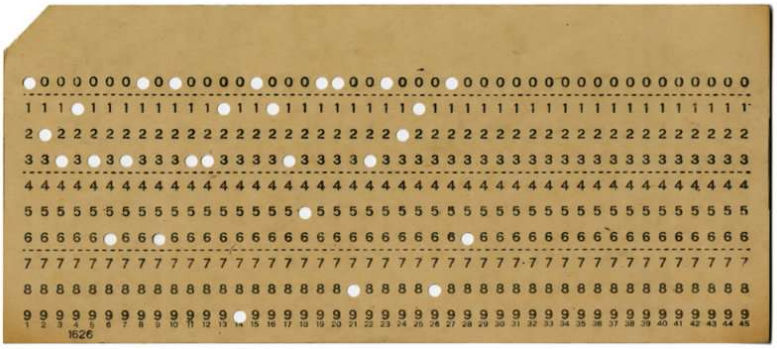
User Input Handling

Paul Haskell

CS112



How do users enter input into a computer?
- keyboard, mouse, touchscreen, data network

Picture shows a computer punch card. People would type in characters on a special machine that would punch holes in the cards. Computers had "punch card readers" that would read big stacks of the cards, which would hold your programs and data. Even your compilers.

These predate me by about 20 years.

Bit of computer history
- First "mechanical adding machine in 1600s. Other interesting mechanical computers included:
  - Programmable looms for weaving patterns into fabric
  - Analog computers for doing integration or solving differential equations
  - Artillery sighting
- First "all-electronic digital computer" maybe ENIAC, 1945
- First PCs about 1980

User Input

`InputStream System.in` – delivers input from keyboard as a "stream"

`class Scanner(InputStream)` - scans the input stream and returns different types of data

A few methods:

- `String Scanner.next()` – returns the next "word" (up to a space)
- `boolean Scanner.nextBoolean()`
- `byte Scanner.nextByte()`
- `double Scanner.nextDouble()`
- `String Scanner.nextLine()` – returns text up to the next End-Of-Line

Talked about input from cmd line. Now from keyboard
We won't dive into streams now. It's an object we can keep reading from.
Need import java.util.Scanner;
IMPORT like in Python. Include modules so you can use them in your software

Reading User Input

The three key lessons for the semester:

- **Design your classes thoughtfully so they are useful and reusable**
- **Spend as much time thinking about testing as you do about your software design; spend as much time testing as you spend coding**
- **Never trust user input**

Reading User Input

A few more useful `Scanner` methods:

- `boolean Scanner.hasNext()` – is there an input word available?
- `boolean Scanner.hasNextInt()` – input int available?
- `boolean Scanner.hasNextFloat()` – input float available?
- `boolean Scanner.hasNextLine()` – is there a line available?

See ReadKeyboard.java
Did the user give you what you asked for?

Scanner

Scanner can read from a `String` or a `File` also

Input Redirection

Instead of actual typed input data, the Operating System let's us pretend to send keyboard input to a program while actually taking it from a file.
So when I'm testing 60 students' homework this week, I'm not actually going to type in the same damn thing 60 times.

Redirection an "Operating System thing", not Java specific

```
% cat myFile
11 22 33 44 55 66

% java ReadKeyboard < myFile
Read an int:  11
Read an int:  22
Read an int:  33
Read an int:  44
Read an int:  55
Read an int:  66
```

It's the "<" character
NO COMMAND LINE ARGUMENTS! The OS handles this before calling the program.

More Robustness

What if we don't have enough input?

Summary:

- **Must validate that TYPE of input is what is expected**
- **Must validate that we actually have input**

Can save output to a file

```
% java ReadCommandLine 11 22 33 44 55 66 > outputFile

% cat outputFile
Read an int:  11
Read an int:  22
Read an int:  33
Read an int:  44
Read an int:  55
Read an int:  66
```

It's the ">" character
Can do both < and >, of course.

System.err

A second object that prints to the screen, for error reporting

```
System.err.println("Your input is bad…and so are you!");
```

When we redirect output, can direct System.out ("stdout") and System.err ("stderr") separately.

GO TRY IT in demo code!

CS112 –
Java
Programming

Wrapper Classes

"Wrapper" classes

`Byte, Short, Integer, Long, Float, Double, Character, Boolean`

- **Class** types that "put a wrapper around" the built-in types

Why use these?

- Sometimes you need a class type e.g. to be stored in Java's advanced storage data structures (arrays, trees, dictionaries, etc)
- Or we want "advanced" functions in the class types, e.g. `Character.isLetter()`, `Character.isDigit()`, `Character.toLowerCase()`

Try this out!

Special capabilities of Wrapper Classes

AUTOBOXING

```
Integer myValue = 1; // how do we mix Integer and int?
void doubleMe(Integer x) { x = x * 2; }
```

Similar to the String operation:

```
String myStr = "Look, no call to new!";
```

AUTOBOXING: automatic casting, but only for the Boxed classes. CANNOT DO THIS WITH YOUR OWN CLASSES! (e.g. Doubler)

But more trickiness!

```
void doubleMe(Integer x) { x = x * 2; }

Integer myValue = 1;
doubleMe(myValue);
System.out.println(myValue); // what does this print?
```

It prints 1

- Why??

The "x = x * 2"; statement does not change the value of our Integer object

It creates a new Integer object and has x (but not myValue) point to it

But more trickiness!

Another example

```
void fixMe(String s) { s = s + " (sometimes)"; }
String myString = "I love Java!";
fixMe(myString);
System.out.println(myString);
```

What does this print?

"I love Java!"

String variables never change values! New String objects are created

But recall…

```
void ProcessNumbers(Doubler ddd) {
   ddd.set(20.0);
}
Doubler myDoubler = new Doubler();  myDoubler.set(1.0);
ProcessNumbers(myDoubler);
System.out.println(myDoubler.get()); // prints 40.0
```

Why?

We programmed `class Doubler` so its stored value can change.

Java's built in classes (String, Byte, Integer, …) are immutable.

You just have to remember this.

Immutable. Cannot change value of an object. CAN create a new object

Characters in Java

Character variables

Used to represent a letter, number or punctuation

…or maybe "control codes"

- tab, newline, backspace, EndOfTransmission, Acknowledge
- Used for communications between computers, not just text on a screen

ASCII character set (LINK)

What is a "character set"?

Ok, but what about…

Spanish? French? Italian? Led to "Latin-1" character set

- German is not a Latin language!

À Á Â Ã Ä Å Æ Ç È É Ê Ë Ñ Ø …

Ok, but what about...

Greek, Cyrillic, Arabic, Hebrew, Chinese, Japanese, Korean, Tibetan, Mongolian, Syriac, Phoenician, Imperial Aramaic, Bengali, Malayalam, Balinese, …

Unicode

Over 140k characters. Including mathematical symbols, emojis, ancient languages, etc

A "living" charset: adding Mayan, Tengwar (Tolkien Elvish). Klingon (Star Trek) was considered and rejected for now

Unicode

Each Unicode character has a "code point" i.e. index (LINK)

Character	Code Point	Description
A	65	Latin capital A
a	97	Latin lowercase a
£	163	British Pound
Λ	955	Greek lowercase lambda
Ц	1062	Cyrillic capital tse
ض	1590	Arabic dad
৪	2538	Bengali digit '4'
∞	8734	infinity
鵺	40898	Han character

All ASCII chars have same code point as in ASCII

Character Sets vs Fonts

Character Set – a set of specified characters that people/programs/etc agree to work with

Code Point – the numerical index for each character in a character set

Font – a particular set of graphical representations for a character set

So Java uses Unicode Code Points?

Not quite…

Character Set – a set of specified characters that people/programs/etc agree to work with

Code Point – the numerical index for each character in a character set

Font – a particular set of graphical representations for a character set

Encoding – a mapping from code point values to compressed encoded values

Why do this?

Unicode Encodings

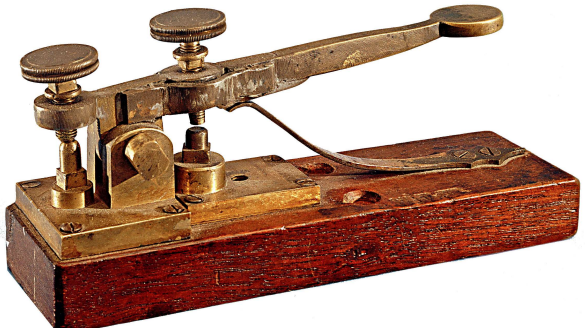
Efficiency

- Unicode code points require 21 bits ➔ 3 bytes per character
- Many common applications (Latin text) only require 1 byte per character

How to compress?

Compression Origins

Morse Code



A	.–	M	––	Y	–.––
B	–...	N	–.	Z	––..
C	–.–.	O	–––	1	.––––
D	–..	P	.––.	2	..–––
E	.	Q	––.–	3	...––
F	..–.	R	.–.	4	....–
G	––.	S	...	5	.....
H	....	T	–	6	–....
I	..	U	..–	7	––...
J	.–––	V	...–	8	–––..
K	–.–	W	.––	9	––––.
L	.–..	X	–..–	0	–––––

You know the distress signal "SOS"? Know why it is the distress signal? B/c Morse Code rep'n is easy pattern: …---…
Morse code used in telegraph. Telegraph probably biggest communication revolution in history, except maybe for printing press
What is Morse code? Patterns of dots and dashes, to represent English letters and digits 0-9
What's the magic? **Common letters have short codes**, uncommon letters have long codes
Avg length = sum (prob(each char) * len(each char))

Unicode Encodings

Two common encodings

- UTF-8: 1, 2, 3, or 4 bytes per character. Used by WWW, Linux Operating Systems (files)
- UTF-16: 2 or 4 bytes per character. Used by Java, Windows OS (files)

UTF-16 examples

Character	Unicode Code Point	UTF-16 Encoding
A	65	65
Z	122	122
쪬 ("CJK" symbol jjoenh)	51 990	51 990
𐡹 (Aramaic ten thousand)	67 679	3 624 066 143
🀙 (Mahjong plum card)	127 010	3 627 867 170

Java includes CONVERSION ROUTINES to translate between UTF-8 (files) and UTF-16 (Java variables)
Most folks agree UTF-16 choice was mistake—was done for **bwd compat** with UTF-2.
Talk about MPEG/AVC/HEVC:
1B MPEG2 decoders, $100 ea, $500 for tech visit. Cost > GDP of Sweden, Norway, Israel, Nigeria, Vietnam, etc

Ok, *now* can we talk about Java code?

Java `char` variables

16 bit

Integer-like

- +, -, /, *
- `char ch = 65;`

Unsigned: values in range 0-65535

Stores UTF-16 encoded value of a Unicode character

But...

65535 is not big enough to represent half of the Unicode characters!

Need two `char` values to represent many Unicode characters

Java does not always handle "big" characters correctly

- `String s = "" + '\u20000';`
- `System.out.println(s.length()); // prints 2 but should be 1`
- `String s2 = s.substring(0, 1); // splits big character in half`

The (somewhat) good news…

- Not many applications need "big" characters (but plenty do).

My computers Java applications (Eclipse, cmd, Notepad++) don't support "big" chars

Tell me about those conversion routines

UTF-16 / UTF-8 conversion

All Java `char` variables store UTF-16 encoded values

- Same for `String` variables

Where could UTF-8 data come from?

- Probably from files
- Possibly from byte arrays, some other software

Many "Reader" classes let you specify encoding of input

```
String s2 = new String(byteArray, StandardCharsets.UTF_8);
InputStreamReader isr = new InputStreamReader(inp, StandardCharsets.UTF_8);
Scanner s = new Scanner(someInputStream, "UTF-8");
```

Sometimes String with name of charset, sometimes a constant representing the charset

InputStream reads bytes and byte arrays
InputStreamReader reads chars and char arrays
Scanner reads ints, doubles, Strings, Objects, etc.

UTF-16 / UTF-8 conversion

Works in other direction too

```
byte[] bArr = stringVar.getBytes(StandardCharsets.UTF_8);
OutputStreamWriter osw = new OutputStreamWriter(outStream, StandardCharsets.UTF_8);
PrintWriter pw = new PrintWriter("filename", "UTF-8");
```

OutputStream writes bytes and byte arrays
OutputStreamWriter writes chars and char arrays
PrintWriter writes ints, doubles, booleans, Strings, Objects, etc.