To the right is a simple class \texttt{IntBox}, an instance of which contains an \texttt{int} value. It could have many more methods, but it has just what we need to explain parametric polymorphism.

Now suppose someone asks us to create a box class to hold a \texttt{char} value instead of an \texttt{int} value. We can copy class \texttt{IntBox} and change the copy to be a \texttt{CharBox}.

Now suppose someone asks us to create a class to hold a \textit{third} kind of value. How many times are we going to have to create a new class, copy, paste, and edit? There must be a better way.

There \textit{is} a better way, called \textit{parametric polymorphism}, and it is implemented in Java using \textit{generics}. As shown to the right, we give class \texttt{Box} a type \texttt{parameter} \texttt{T} enclosed in “<” and “>”, shown in red.

Then, field \texttt{contents} has type \texttt{T}, parameter \texttt{t} of method \texttt{put} has type \texttt{T}, and the return type of method \texttt{get} is \texttt{T}. Also, when a new \texttt{Box} object is created, the value in it will be the default value for \texttt{T}.

To create a \texttt{Box} object that can contain an \texttt{Integer} value (pointer to an \texttt{Integer} object) and a \texttt{Box} object that can contain a \texttt{Character} value, use these two statements:

- \texttt{Box\<Integer> bi= new Box \<>();}  
- \texttt{Box \<Character> bc= new Box \<>();} 

In the first statement, \texttt{Integer} is the \textit{type argument} given for type parameter \texttt{T}; similarly for \texttt{Character} in the second statement.

Think of this as giving us objects of classes like

- Class \texttt{IntBox} above, except that wrapper class \texttt{Integer} is used in place of \texttt{int}.  
- Class \texttt{CharBox} above, except that wrapper class \texttt{Character} is used in place of \texttt{type char}.

The wrapper classes have to be used because type arguments may only be class types and never primitive types. That’s OK; Java will automatically box and unbox between primitive type values and their wrapper classes. We don’t have to be concerned too much with that.

This, then, is \textit{parametric polymorphism}. The use of a type parameter allows us to have many boxes, which differ only in the types of values they can hold.

Note the following. We generally write the new-expression in the assignment to \texttt{bi} above as shown below, with \texttt{Integer} between the < and >. But in this statement we can omit \texttt{Integer} because Java can infer what should be between < and > from the context.

\texttt{Box\<Integer> bi= new Box \<>();} 

We have shown the simplest use of \textit{generics} in Java. There is much more. Look at the entry for generics in \textit{JavaHyperText}.  

\begin{verbatim}
/** object contains an int. */
public class IntBox {  
    private int contents;
/** Constr: box with 0. */
public IntBox() {}  
/** Put \texttt{t} into the box */
public void put(int \texttt{t}) {  
    contents= \texttt{t};
}  
/** Return contents. */
public int get() {  
    return contents;
}  
}

/** object contains a char. */
public class CharBox {  
    private char contents;
/** Constr: box with \texttt{'u0000'}. */
public CharBox() {}  
/** Put \texttt{\texttt{t}} into the box */
public void put(char \texttt{t}) {  
    contents= \texttt{t};
}  
/** Return contents. */
public char get() {  
    return contents;
}  
}

/** object contains a \texttt{T}. */
public class Box\<\texttt{T}> {  
    private \texttt{T} contents;
/** Constr: box with default value of \texttt{T}. */
public Box\<\texttt{T}>() {}  
/** Put \texttt{\texttt{t}} into the box */
public void put(\texttt{T} \texttt{t}) {  
    contents= \texttt{t};
}  
/** Return contents. */
public \texttt{T} get() {  
    return contents;
}  
}
\end{verbatim}